

# Integer Programming for Media Streams Planning Problem\*

Pavel Troubil and Hana Rudová

Faculty of Informatics, Masaryk University  
Botanická 68a  
60200 Brno, Czech Republic  
pavel@ics.muni.cz, hanka@fi.muni.cz

**Abstract.** Continually increasing demands for high-quality videoconferencing have brought a problem of fully automated environment setup. A media streams planning problem forms an important part of this issue. As the multimedia streams are extremely bandwidth-demanding, their transmission has to be planned with respect to available capacities of network links and the plan also needs to be optimal in terms of data transfer latencies. This paper presents an integer programming solution of the problem and its implementation. The implementation achieved very promising results in performance-evaluating measurements. Compared to previous constraint-based solver, it is capable of finding optimal solution significantly faster, allowing for real-time planning of larger problem instances.

## 1 Introduction

Modern computer networks allowing high-bandwidth transmissions have become more and more widespread recently. Their increasing availability heavily supports deployment and user-adoption of advanced collaborative environments. These environments frequently require transmission of very high-bandwidth data streams. Smoothness and enjoyability of synchronous remote collaboration also crucially depends on a low-latency transmission of the data streams.

Setting up an advanced collaborative environment might be a difficult and tedious task, probably undesirably hard for end-users. The setup often comprises configuring of potentially high number of individual components (e. g., data producers, processors, distributors, or consumers), and also data distribution paths in a network. As a bandwidth needed for transmission of the data streams is frequently close to capacities of state-of-the-art backbone links, finding out correct and latency-minimal distribution paths also becomes a very complicated task.

In order to automate the process of the environment setup, the CoUniverse framework has been proposed in [5]. The problem of deciding the data distribution paths has been formally defined as a media streams planning problem

---

\* This research is supported by the Ministry of Education, Youth and Sports of the Czech Republic under the project 0021622419.

(MSPP) in [3]. The MSPP is a network optimization problem close to a multi-commodity network flows problem [1]. A survey of network optimization problems can be found in [9]. The MSPP is also strongly related to a multicast routing problem [6] for multiple multicast groups (called multicast packing problem). Unfortunately the multicast service is not proper for our purpose since it is not continuously deployed over the whole Internet, and lacks performance needed for high-speed transmissions. Still methodologies applied to solve this problem can provide an inspiration for our work. Heuristics for this problem were presented in [2, 10, 8] and an optimal solution using linear programming is known for single multicast problem even if data streams are allowed to split [4].

A constraint-programming based solver of the MSPP has been implemented in the CoUniverse [3]. This solver is only capable of solving medium-sized instances of the problem quickly enough, i. e., in a few seconds. With the aim to solve larger problem instances, we propose a new solver based on integer programming. We have rewritten the previous constraint programming model to a form of an integer programme, reformulating some improper disjunctive constraints. The IP model was implemented in the CoUniverse framework and evaluated in terms of performance. Results of the evaluation show that our solver is capable of solving larger instances of the problem for real-world network topologies, e. g., those used for distributed lectures.

## 2 Problem description

Generally said, the media streams planning problem is a network optimization problem of computing data distribution paths in a network. Given a topology of the network, a set of data sources, and destinations for the data delivery, the goal is to find a distribution tree for each of the data sources. The data distribution paths are required to be optimal in terms of overall transmission latency. Media streaming network applications usually require all data transmissions to occur at the same time, e. g., in a videoconference. Hence no temporal parameters are considered in the MSPP. In order to handle changes in the network (e. g., link outage), the CoUniverse monitors the environment and may invoke a replanning, i. e., a new call of the solver.

Main entity of the network is a *node*  $v \in V$ . Since we abstract from physical network devices such as switches or routers, the nodes represent computers (servers as well workstations), each of them running one or possibly more *applications*. The applications either produce a data stream (called *producers*,  $p \in P$ ), consume the stream (*consumers*,  $c \in C$ ), or distribute it (*distributors*,  $d \in D$ ), possibly creating multiple copies of received data. The applications are capable of processing one data stream at most, i. e., a single application can neither produce, consume, nor distribute more than one stream. If an application  $a \in P \cup D \cup C$  is running on a node  $v$ , we write  $a \in v$ . The distributors are server applications providing a multicast functionality on an application layer of the ISO/OSI model. UDP packet reflectors and Active Elements are examples of such applications.

If there is a distributor running on a network node, no other application is allowed to run there, i. e., no producers and/or consumers may run on that node, neither may any other distributor. If there is no distributor on a node, there may be running several producers and consumers together. Their number is not limited as long as they all process different streams. Yet, this model targets primarily on CPU-intensive multimedia applications that might cause overload of the node when run simultaneously. Nodes are organized into *sites*, which generally represent geographical collocation of the nodes. There are typically several nodes at a site, each of them running a single application.

Each network node has several network *interfaces*  $i \in I$  configured. We denote by  $i \in v$  that the interface  $i$  is configured on the node  $v$ . An interface has a limited transmission capacity  $capI(i)$ . Every interface belongs to a *subnetwork*. We consider each interface reachable from all other interfaces in the same subnetwork and vice versa.

In other words, there is a *link*  $l \in L$  between each ordered pair of interfaces, i. e., it is strictly directional. Links together with nodes form a directed graph  $N = (V, L)$  corresponding to the underlying network. The links represent a network infrastructure which facilitates the data transfer between the pair of network nodes. Although links formally interconnect interfaces, we often speak about them as connecting *nodes*. Naturally, a link between interfaces  $(i_1, i_2)$  connects nodes  $(v_1, v_2)$  if and only if  $i_1 \in v_1 \wedge i_2 \in v_2$ . Source and target nodes of a link  $l$  are denoted  $begin(l)$  and  $end(l)$ , respectively. We also denote a set of links connected to an interface  $i$  as  $links(i)$ .

For an *application*  $a \in P \cup D \cup C$  on a *node*  $v$ , we define  $inlinks(a)$  as a set of links ending in the *node*  $v$ , i. e.,  $l \in inlinks(a) \iff a \in end(l)$ . We extend this notation on a set of applications  $A \subseteq P \cup D \cup C$ :  $l \in inlinks(A) \iff a \in end(l)$  for some  $a \in A$ . We also define  $outlinks(a)$  and  $outlinks(A)$  analogously.

Each network link  $l$  has two attributes: its maximum capacity  $cap(l)$  and transfer delay  $latency(l)$ . Since the links do not represent physical topology of a network, several links in our model might share one physical network link. Consequently, whole bandwidth of  $cap(l)$  might not be actually available for transmission. A static configuration of *capacity* of the links is augmented by a real-time monitoring in the CoUniverse, similarly as *latency* of the links needs to be also monitored.

The goal of media streams planning is to determine paths for the distribution of the data *streams*  $(s \in S)$ . We use the term *stream* since motivation for the MSPP lies in continuous multimedia transmissions. We denote a bandwidth required for transmission of the stream  $s$  as  $bw(s)$ . Each stream  $s$  is produced by a single application  $producer(s) \in P$  and is required to be delivered to a set of consumers  $consumers(s) \subseteq C$ . There is exactly one producer of the stream  $s$  and at least one consumer of the stream, i. e.,  $consumers(s) \neq \emptyset$ . Transmission of the streams and possibly creation of multiple data copies is performed by the distributors. More precisely, each stream is transferred from a producer to a set of consumers by a communication tree with the producer in its root, consumers at leafs, and media distributors at internal nodes. Therefore, the problem may

be considered as a tree placement [3] in contrast to classical path placement [9] where no data multiplication is processed. On the other hand, all packets of each stream from its producer to a single consumer have to be transferred along the same path. If the packets would be transferred along more than one path, unfavourable reordering of the packets would occur due to different latencies of the paths. The distributors are therefore not allowed to send data between any producer—consumer pair from a node through more than one link (e.g., for load balancing purposes).

### 3 Integer Programming Model

For each stream  $s$  and network link  $l$ , we introduce a binary decision variable  $x_{s,l}$ , further denoted as *streamlink*. The streamlink  $x_{s,l}$  equals to 1 if the stream  $s$  is transmitted over the link  $l$ , otherwise it corresponds to 0.

We also call a streamlink  $x_{s,l}$  *active* if and only if  $x_{s,l} = 1$ , or *inactive* otherwise. Since our aim is to minimize overall transmission latency, we formulate the objective function as a sum of latencies of all active streamlinks.

$$\min \sum_{s \in S} \sum_{l \in L} x_{s,l} \cdot \text{latency}(l)$$

The three following constraints implement network capacity limitations. By constraint (1), it is not allowed to transfer a stream  $s$  through a link  $l$  of insufficient capacity. We set the decision variable directly to zero when the link does not have sufficient capacity for transmission of the stream. This constraint is a redundant one and follows from the consequent constraint (dependent on decision variables). Constraint (2) guarantees that the capacity of a link  $l$  cannot be exceeded by a total bandwidth of the streams transferred over the link. Constraint (3) states that the capacity of an interface  $i$  cannot be exceeded by a total bandwidth of the streams transferred over all links connected to this interface. The presented variant of the constraint is used for interfaces which do not support full duplex. On full duplex interfaces, incoming and outgoing links are treated separately, since they do not interfere with each other.

$$x_{s,l} = 0 \quad \forall s \in S \forall l \in L \text{ s.t. } bw(s) > cap(l) \quad (1)$$

$$\sum_{s \in S} bw(s) \cdot x_{s,l} \leq cap(l) \quad \forall l \in L \quad (2)$$

$$\sum_{s \in S} \sum_{l \in \text{links}(i)} bw(s) \cdot x_{s,l} \leq capI(i) \quad \forall i \in I \quad (3)$$

A network node cannot send a stream  $s$  over arbitrary outgoing links. To allow transmission of the stream  $s$  over a link  $l$ , either consumer of this stream or a distributor must reside on the target node of the link  $l$  (4). A similar rule holds for receiving of the stream. To allow transmission of the stream  $s$  over

a link  $l$ , either producer of this stream or a distributor must reside on the source node of the link  $l$  (see constraint (5)).

$$x_{s,l} = 0 \quad \forall s \in S \forall l \notin \text{inlinks}(D \cup \text{consumers}(s)) \quad (4)$$

$$x_{s,l} = 0 \quad \forall s \in S \forall l \notin \text{outlinks}(\{\text{producer}(s)\} \cup D) \quad (5)$$

Each producer is capable of producing a stream and sending it to another network node and does not have any additional data distribution capabilities. Consequently any producer is required to send the stream over exactly one link.

$$\sum_{l \in \text{outlinks}(\text{producer}(s))} x_{s,l} = 1 \quad \forall s \in S \quad (6)$$

If there is more than one consumer of a stream  $s$ , its producer is not allowed to send the stream directly to any consumer. We disable all direct links from the producer to all consumers of the stream  $s$ . For each stream  $s$  such that  $\|\text{consumers}(s)\| > 1$ , we introduce the constraint (7). This constraint is redundant, as the constraint (4) is sufficient to force inactivity of these links when combined with the other constraints.

$$x_{s,l} = 0 \quad \forall l \in L \quad \text{s.t.} \quad l \in \text{outlinks}(\text{producer}(s)) \cap \text{inlinks}(\text{consumers}(s)) \quad (7)$$

Each producer is required to send the data to each consumer along a single path. This means that any consumer does not need to receive a stream from more than one node unless there is some redundant transmission. The consumer is therefore required to receive the stream over exactly one link.

$$\sum_{l \in \text{inlinks}(c)} x_{s,l} = 1 \quad \forall s \in S \forall c \in \text{consumers}(s) \quad (8)$$

The following constraints are aimed to make each stream  $s$  transferred along a tree rooted at a node where  $\text{producer}(s)$  resides. A distributor  $d$  is allowed to distribute one stream at most, and the stream has to be transferred to any network node only by a single network link. In addition, each node containing a distributor cannot contain any other application. Following these rules, there may be at most one active streamlink incoming in the distributor's node (see (9)). Constraints (10) and (11) guarantee that a stream  $s$  is sent by a distributor  $d$  if and only if it is also received by  $d$ . Contrary to the constraint (9), they have to be formulated on a per-stream basis. In this case, summing all streamlinks would allow the distributor to send further arbitrary stream no matter which stream it receives. Since corresponding constraints of the constraint programming model [3] are not suitable for the integer programming due to their improper statement with disjunctions, their reformulation was necessary. Constraint (10) states that there are not less active outgoing links than active incoming links, i. e., the distributor  $d$  is forced to forward an incoming stream. Next, a distributor  $d$  is not allowed to forward any stream it does not receive. As the distributor may only distribute a single stream,  $\|\text{outlinks}(d)\|$  corresponds

to the maximum possible number of streamlinks over which the distributor may send any data (11).

$$\sum_{s \in S} \sum_{l \in \text{inlinks}(d)} x_{s,l} \leq 1 \quad \forall d \in D \quad (9)$$

$$\sum_{l \in \text{inlinks}(d)} x_{s,l} \leq \sum_{l \in \text{outlinks}(d)} x_{s,l} \quad \forall s \in S \quad \forall d \in D \quad (10)$$

$$\|\text{outlinks}(d)\| \cdot \sum_{l \in \text{inlinks}(d)} x_{s,l} \geq \sum_{l \in \text{outlinks}(d)} x_{s,l} \quad \forall s \in S \quad \forall d \in D \quad (11)$$

A distribution tree of each stream  $s$  is limited in size by the number of consumers of  $s$  and the number of available distributors. There are three redundant constraints (12), (13), and (14) to support that. First, the distribution tree may include at most  $1 + \|D\| + \|\text{consumers}(s)\|$  nodes. Since cycles among nodes are not allowed (see constraint (15)), maximum number of links over which the stream  $s$  may be transferred is equal to  $\|D\| + \|\text{consumers}(s)\|$  (see (12)). Next, if there is only a single consumer of a stream  $s$ , one link may be sufficient for transmission (13). Otherwise, the stream has to be transmitted through at least one distributor, setting the minimal number of needed links to  $1 + \|\text{consumers}(s)\|$  (see (14)).

$$\sum_{l \in L} x_{s,l} \leq \|D\| + \|\text{consumers}(s)\| \quad \forall s \in S \quad (12)$$

$$\sum_{l \in L} x_{s,l} \geq 1 \quad \forall s \in S \text{ s. t. } \|\text{consumers}(s)\| = 1 \quad (13)$$

$$\sum_{l \in L} x_{s,l} \geq 1 + \|\text{consumers}(s)\| \quad \forall s \in S \text{ s. t. } \|\text{consumers}(s)\| > 1 \quad (14)$$

The last constraint eliminates cycles that might occur among nodes with distributors. The previous constraints allow existence of a cycle in which each distributor may receive a stream from another distributor, potentially forwarding the stream on a path to one or more consumers. The cycle-avoidance constraints are derived from the graph theory results. If a graph with  $k$  vertices has more than  $k - 1$  edges, there is a cycle in the graph. Further, if there is not a cycle in any subgraph of a graph then the graph does not contain any cycle either. We denote the number of distributors  $\|D\|$  as  $n$ . To avoid cycles among the distributors, we put an upper bound on the number of edges in each  $k$ -tuple of distributors for  $2 \leq k \leq n$ . For  $n$  distributors, there are  $\binom{n}{k}$  subsets of  $k$  elements in total, i. e.,  $k$ -tuples of distributors. We denote a set of all distributor  $k$ -tuples as  $D_k$ , and its  $i$ -th member as  $D_k(i)$ . For each stream  $s$  and each  $k$ -tuple of distributors, we introduce the following constraint.

$$\sum_{\substack{j_1, j_2 \in D_k(i) \wedge \\ v_{j_1} = \text{begin}(l) \wedge v_{j_2} = \text{end}(l)}} x_{s,l} \leq k - 1 \quad \forall s \in S \quad \forall k \in \{2, \dots, n\} \quad \forall i \in \{1, \dots, \binom{n}{k}\} \quad (15)$$

This formulation is similar to cycle-avoidance constraints in subtour formulation of cycle elimination in the travelling salesman problem (TSP) [7]. The main difference is that we need to apply the constraint even for cycles containing more than  $n/2$  nodes. In the TSP, occurrence of larger cycle would necessarily enforce occurrence of another cycle with less than  $n/2$  nodes; yet, this assumption does not hold in the MSPP.

*Redundant Constraints* The redundant constraints were kept in the model although they do not strengthen the formulation. On the other hand, they may improve performance of the MIP solvers significantly. Evaluation of their influence on the solver performance will be part of our follow-up work.

## 4 Evaluation and Results

We modified an MSPP solving module in the CoUniverse to implement the integer programming model. The module is written in the Java programming language and uses the Gurobi Optimizer<sup>1</sup> version 3.0.0 as a backend MIP solver.

Three topologies simulating typical data distribution patterns in advanced collaborative environments were chosen for evaluation of the solver performance:

- (a)  $1:n$  topology: one site  $si$  transmits a stream to all other sites through a single distributor, and each of the other sites transmits a stream back to  $si$ . Further denoted  $1:n-s$ .
- (b)  $1:n$  topology: it is similar to the previous one with an exception of higher number of distributors — there is one for each site except  $si$ . The topology is further denoted  $1:n-r$ .
- (c)  $m:n$  topology: each site transmits a stream to all other sites through its own distributor(s).

These topologies were taken from [3] to compare with their results (see this paper for more detailed description of the topologies and their relation to real-world problems).

All measurements were performed on a PC equipped with Intel Xeon 5160 @ 3.0 GHz quadcore processor and 6 GB RAM, running Linux 2.6.22-17 and Java SDK 1.6.0 in a virtualized Xen environment. Options for `java` were set to `-server -da -dsa`. Each measurement was continuously repeated 20 times, and only the last 5 runs were taken into account. A measurement timer had 4 ms resolution. We did not limit the number of processor cores available to the Gurobi optimizer. Unfortunately, we did not observe any significant performance differences when compared to single-thread runs.

Numbers of nodes and links in instances of the topologies (parametrized by number of sites) are shown in Table 1. The number of links is presented after an elimination process (same as the one applied in [3]).

Results of the performance measurements are shown in Table 2. The measured times (in milliseconds) are split in two parts: preparation (creation of

<sup>1</sup> <http://www.gurobi.com>

**Table 1.** Parameters of topologies used for performance measurement

Topology	$1:n-s-2$	$1:n-s-4$	$1:n-s-8$	$1:n-s-16$	$1:n-s-32$	
Nodes	5	11	23	47	95	
Edges	10	44	184	752	3,040	
Topology	$1:n-r-2$	$1:n-r-4$	$1:n-r-6$	$1:n-r-8$	$1:n-r-10$	$1:n-r-12$
Nodes	5	13	21	29	37	45
Edges	10	78	210	406	666	990
Topology	$m:n-2$	$m:n-3$	$m:n-4$	$m:n-5$	$m:n-6$	$m:n-7$
Nodes	6	12	20	30	42	56
Edges	18	60	140	270	462	728

**Table 2.** Times in milliseconds required to solve the topologies

Topology	$1:n-s-2$	$1:n-s-4$	$1:n-s-8$	$1:n-s-16$	$1:n-s-32$	
Preparation	$4.0 \pm 0$	$5.6 \pm 2$	$51 \pm 2$	$950 \pm 30$	$24,000 \pm 200$	
Optimization	$< 4.0$	$< 4.0$	$2.4 \pm 2$	$33 \pm 2$	$370 \pm 5$	
Topology	$1:n-r-2$	$1:n-r-4$	$1:n-r-6$	$1:n-r-8$	$1:n-r-10$	$1:n-r-12$
Preparation	$4.0 \pm 0$	$9 \pm 2$	$39 \pm 2$	$140 \pm 6$	$410 \pm 8$	$1,300 \pm 20$
Optimization	$< 4.0$	$5 \pm 2$	$17 \pm 3$	$120 \pm 2$	$970 \pm 10$	$7,900 \pm 18$
Topology	$m:n-2$	$m:n-3$	$m:n-4$	$m:n-5$	$m:n-6$	$m:n-7$
Preparation	$< 4.0$	$5.6 \pm 2$	$16 \pm 0$	$46 \pm 3$	$120 \pm 5$	$270 \pm 2$
Optimization	$< 4.0$	$< 4.0$	$7.2 \pm 2$	$18 \pm 2$	$39 \pm 3$	$100 \pm 2$

variables and constraints, elimination of the links), and optimization, which is performed by the backend solver solely. The largest instances of each topology represent current limitation of the solver for real-time application. In case of the  $1:n-s$  topology, the preparation phase is the bottleneck. The  $1:n-r-12$  topology is already above interactivity requirements. Steep growth in the computation time is primarily caused by corresponding steep increase in the number of cycle-avoidance constraints (15). These were needed due to the increasing number of distributors. Similarly, the  $m:n$  topology can be solved for seven sites at most, as the  $m:n-8$  topology requires many more distributors and consequently also cycle-avoidance constraints.

Compared to the previous CP-based solver [3], limitation in solving the  $1:n-s$  topologies stays roughly the same. However, most of the time is spent by the preparation phase, not by the IP solving. We will further pursue this issue to improve performance of the preparation phase for larger instances. The IP solver allows to solve the  $1:n-r$  topology for ten sites in real-time. This is a significant improvement compared to the CP-based solver, which allows for five sites at most. We also achieved an improvement for the  $m:n$  topologies, shifting from five to seven sites. The results might possibly be improved by a different formulation of cycle-avoidance.

## 5 Conclusions

Aiming to develop faster solver for the media streams planning problem, we presented the solution based on the integer programming model. Measured performance of the new solver shows promising results, shifting size of the problem instances which can be solved in real-time.

In our future work, we will evaluate performance of the solver more elaborately. We will also evaluate influence of the redundant constraints on performance of the backend solver. Further, we will explore another formulations of the cycle-avoidance constraints and the problem as a whole. Finally, we intend to explore various problem extensions to consider more general problems.

## References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, NJ, 1993.
2. Shiwen Chen, Oktay Günlük, and Bülent Yener. The multicast packing problem. *IEEE/ACM Transactions on Networking*, 8(3):311–318, 2000.
3. Petr Holub, Miloš Liška, and Hana Rudová. Data transfer planning with tree placement for collaborative environments, 2010. Under revision in *Constraints*.
4. Ayaz Isazadeh and Mohsen Heydarian. Optimal multicast multichannel routing in computer networks. *Computer Communications*, 31(17):4149 – 4161, 2008.
5. Miloš Liška and Petr Holub. CoUniverse: Framework for building self-organizing collaborative environments using extreme-bandwidth media applications. In *Euro-Par 2008 Workshops – Parallel Processing*, volume 5415 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2008.
6. Carlos A. S. Oliveira and Panos M. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research*, 32(8):1953 – 1981, 2005.
7. Gábor Pataki. Teaching integer programming formulations using the traveling salesman problem. *SIAM Review*, 45:116–123, 2003.
8. Luca Sanna Randaccio and Luigi Atzori. Group multicast routing problem: A genetic algorithms based approach. *Computer Networks*, 51(14):3989–4004, 2007.
9. Helmut Simonis. Constraint applications in networks. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 875–903. Elsevier, 2006.
10. Chu-Fu Wang, Chun-Teng Liang, and Rong-Hong Jan. Heuristic algorithms for packing of multiple-group multicasting. *Computers & Operations Research*, 29(7):905–924, 2002.