Multipoint-to-multipoint network communication

Philip Kilby
NICTA and Australian National University

Desmond Lun
University of South Australia

Giang Nguyen
University of South Australia

1 Introduction

Multi-user virtual environments are becoming popular in a number of areas. Complex virtual environments are becoming available, such as Massively Multiplayer Online Role Playing Games (MMORPG) (e.g., *World of WarCraft* and *Travian*) and social networks (e.g. *Second Life*). The number of participants in these environments can be hundreds, thousands or even tens of thousands of people. Within such a virtual environment, the users, typically represented by their *avatars*, can freely interact with other avatars or virtual objects on the network. In many situations, a user might want to talk to one or more users on the network. Then the user has to send his or her voice along the network to the intended recipient(s). Similarly, users might also wish to broadcast their own live videos, images or other content such as video clips, audio files or virtual objects created by themselves.

Currently, most MMORPG employ the *client-server model*, where the server has complete information about the virtual world, and facilitates the dissemination of multimedia information amongst the users [5]. For example, if Alice wants to communicate with Bob, the information will be sent along the network to the server, which will then pass the information along the network to Bob. It is obvious that maintaining the server presents a huge cost factor, especially as the environment grows larger. An alternative model, currently the focus of study by The University of Wollongong's ICT Research Institute, employs a *peer-to-peer model*, which eliminates the need for a server. In this case, Alice can send information along the network to Bob, without going through a server first.

The peer-to-peer topology is a kind of "user-pays" system – in effect users provide the infrastructure. When an underlying network, such as the world-wide web, is used for physical transport, a virtual network is overlaid. Messages are routed through the virtual network, using only a limited part of the physical network. The virtual network is made up arcs linking nodes representing each game participant. Each participant can act as *source*, *relay*, and *sink* of information. The majority of network traffic is therefore passed through the participants themselves. The construction of this virtual peer-to-peer network is the focus of the problem brought to MISG. It is worth noting that peer-to-peer networks have many successful applications, including file sharing (e.g. *eMule*), streaming (e.g. *Pandora*) and internet phones (e.g. *Skype*).

The design and construction of peer-to-peer networks should take into account three main factors. First, most virtual topologies change quickly and continuously: users join the network, move from one location to another or drop out completely. Second, there is limited capability: due to bandwidth capacity, each user can only be connected directly to a certain number of people. Finally, time constraints should be observed: a piece of information has to be sent from one user to another within a desired amount of time [4]. These time constraints are dependent on the distance between users in the virtual network. The further apart two users are in the virtual world, the more tolerant they are of delays in multimedia delivery. Peer-to-peer networks should also continuously reconfigure to accommodate the ever changing nature of the virtual world.

In Section 2, we describe the problem and introduce notation. In Section 3, we propose a feasibility integer linear program (ILP) as an exact formulation of the problem, report preliminary numerical results on small-size problems and formulate a Lagrangian relaxation approach to the original ILP. We explain an heuristic approach to the problem in Section 4, and detail our implementation and numerical results in Section 5.

2 Problem description and notation

The problem can be formally described as follows. Consider a complete network $G(V, E)$. V is a set of nodes, with $v \in V$ representing a user on the network. E is a set of arcs, $(u, v) \in E$ representing a direct connection (via the underlying network) between two users. Associated with each arc (u, v) is a *direct cost* c_{uv} which is the amount of time it takes to physically transfer a piece of information directly from one user to another (direct transmission). These costs reflect the relative position of the nodes in the physical world. We assume the triangle inequality for these costs, that is, the direct arc between any two nodes has no greater cost than any other, indirect, path between the nodes. Mathematically, if u, v and t are mutually adjacent, then $c_{ut} + c_{tv} \geq c_{uv}\ \forall t$. Note that costs c_{ij} are not necessarily symmetric – i.e. c_{ij} is not necessarily equal to c_{ji}.

There is a *time limit* k_{uv} determined for each arc (u, v). This is a *time constraint* signifying the desired maximum time for communication between two users. These costs reflect the relative position of the two users in the *virtual* world. For any two users, the time limit k_{uv} is at least the cost of the direct arc c_{uv} between them: $k_{uv} \geq c_{uv}$ for all $(u, v) \in E$. In current simulations carried out by ICT Research Institute, there are no time constraints for 70 percent of pair-wise distances (because they are too far apart in the virtual world to interact). Finally, there is an *upper bound* m_i for the *degree* $d(i)$ of each node $i \in G$, that is, a limit of the number of people to whom a user can directly connect. The simulated upper bounds generated by the ICT Research Institute can vary from 3 to 10 people.

We wish to determine *a spanning subgraph* H, which is a virtual network that connects all users. The spanning subgraph should observe all degree constraints on nodes and time constraints on paths between pairs of users. However, if all constraints cannot be met, the degree constraints are seen to be more important. That is, it is permissible to exceed some desired time limits. In that case, the objective is to minimise either the number of such violated constraints, or the sum of violations.

For nodes i, j where $(i, j) \in E$ the time for direct transmission from i to j is c_{ij}. We wish to choose a set $H \subset E$ of arcs to use in the virtual network. Let $n_H(i)$ be the number of neighbours of node i in the arc set H. We require $n_V(i) \leq m_i$. Let $c_H(i,j)$ be the shortest time to send a message from i to j using the arcs in H. We require that $c_H(i,j) \leq k_{ij}$.

As the virtual topology changes frequently, this spanning subgraph should be continuously modified to take into account updates in the network. This could be done in two different ways: either a fast algorithm that can be run in real time periodically with high frequency, or an optimal algorithm that is run once then updated with reduced computational time to accommodate virtual topology changes. Assuming that each user has reasonably complete information of the virtual topology, the ideal algorithm is a *distributed algorithm* that runs independently in each user, then partial solutions from each user can be combined to obtain an overall solution for the network.

3 Feasibility integer linear program

3.1 Exact Formulation

To start with, we have formulated the problem as a *feasibility integer linear program* (ILP):

$$\sum_{j|(i,j)\in E} x_{ij}^{(u,v)} - \sum_{j|(j,i)\in E} x_{ji}^{(u,v)} = \begin{cases} 1, & i=u \\ -1, & i=v \\ 0, & \text{otherwise}, \end{cases} \quad \text{for all } i,u,v \in V, \tag{1}$$

$$z_{ij} \geq x_{ij}^{(u,v)}, \quad \text{for all } (i,j)\in E, u,v \in V, \tag{2}$$

$$\sum_{j|(i,j)\in E} z_{ij} \leq m_i, \quad \text{for all } i \in V, \tag{3}$$

$$\sum_{(i,j)\in E} c_{ij} x_{ij}^{(u,v)} \leq k_{uv}, \quad \text{for all } u,v \in V, \tag{4}$$

$$x_{ij}^{(u,v)}, z_{ij} \in \{0,1\}, \quad \text{for all } (i,j)\in E, u,v,\in V. \tag{5}$$

In this ILP, decision variables z_{ij} correspond to arcs $(i,j) \in E$. If a decision variable z_{ij} is 1 in a solution, then the corresponding arc (i,j) is included in the spanning network H, and 0 otherwise. There are arc-path variables x_{ij}^{uv}, each of which determines whether an arc $(i,j) \in E$ is included in a path (u,v) between two users, and these variables are also binary.

The ILP has five sets of constraints. Constraint set (1) ensures that information is preserved in the network. That is, along every path (u,v) between two users u and v, all media content goes into a user on the path has to come out from that user (relayer), unless that user is at the beginning of the path (source) or at the end of the path (sink). Constraint set (2) (arc-path constraints) specifies that if an arc (i,j) is in an intended path between two users, then it is included in the solution corresponding to our desired spanning network H. The third and fourth sets of constraints correspond to upper bounds on degrees of nodes and cost limits of pair-wise paths, respectively. Finally, the fifth set of constraints (5) restricts arc-path variables x_{ij}^{uv} and decision variables z_{ij} to binary values only. As there is no objective function, this is a feasibility program and any solution that satisfies these five sets of constraints determines a feasible spanning network that we are looking for. A potential extra constraint for this ILP is that the decision variables z_{ij} have to be symmetric, that is, $z_{ij} = z_{ji}$ for all i, j, in the final solution. This constraint is equivalent to ensuring that every direct link between two users, if exists, has to be bidirectional.

If a network has n users, then the total number of decision variables and other variables is of order n^4. It is clear that the number of variables quickly grows as the network size increases. This makes the ILP quickly becomes computationally expensive or even intractable for large-size instances. However, it provides a benchmark for heuristic algorithms developed for the problem.

We have relaxed our ILP by partially removing the last set of constraints, that is, the arc-path variables are no longer restricted to binary values. This results in an ILP that is easier to solve, but requires a slight difference in solution interpretation and subsequent network construction. In a solution of the original ILP, if Alice sends Bob a file, this file is sent as a whole along the network via exactly one path. With the employed relaxation, in a solution of the resulting ILP, this file might be split into smaller pieces, each piece to be sent along a different path, and these pieces are regrouped to form the original file at Bob's location.

We have also introduced extra variables y_{uv} corresponding to the amount of violation of each cost limit and modified constraints (4) to be

$$\sum_{(i,j)\in E} c_{ij}x_{ij}^{(u,v)} - y_{uv} \quad \leq \quad k_{uv}, \qquad\qquad \text{for all } u, v \in V. \tag{6}$$

These violation variables gives rise to an objective function, which minimises the total amount of violation of cost limits

$$\min \sum_{u,v\in V} y_{uv}.$$

If this objective function has a value of zero, then we have found a solution that satisfies all constraints, including ones on cost limits of pair-wise paths. This modification is to cater for situations where no feasible solution can be found. In these cases, it gives us a "good-enough" solution and a measure on how close to feasibility this solution is, while still observing the degree constraints.

3.2 Numerical results on small problems

Using the ILP approach described in the previous section, we have successfully solved toy examples using the optimisation solver MILP_CPLEX. These toy examples have complete 10-node networks where arc costs, cost limits and degree constraints are generated randomly and appropriately, using the problem generator described in Section 5. Table 3.2 gives the results for 10 10-node problems.

3.3 Lagrangian relaxation

One way to approach the solution of the original problem is via Lagrangian Relaxation. The following model relaxes the maximum time constraints.

$$\begin{aligned}
&\max_{\lambda,\mu\geq 0} L(\lambda,\mu), \quad \text{where} \\
&L(\lambda,\mu) = \min_{x,z>0} \sum_{u,v}\sum_{i,j} \mu_{ij}^{(u,v)}(x_{ij}^{(u,v)} - z_{ij}) + \sum_{u,v} \lambda_{u,v}\left(\sum_{ij} c_{ij}x_{ij}^{(u,v)} - k_{uv}\right) \\
&\text{s.t. } (1), (3) \text{ and } (5)
\end{aligned}$$

Problem	Feasible solution found	Time (seconds)
#1	Yes	7
#2	Yes	5.9
#3	Yes	5.9
#4	Yes	5.9
#5	Yes	5.9
#6	Yes	5.9
#7	Yes	5.9
#8	Yes	5.9
#9	Yes	5.8
#10	Yes	5.9

Table 1: Results for 10-node networks, using the ILP model and the optimisation solver MILP-CPLEX, on an Intel Pentium 4, 3.2 GHz, Gentoo Linux, 2GB RAM

$$\begin{aligned}
&= \min_{x,z>0} \sum_{u,v} \sum_{i,j} (\mu_{ij}^{(u,v)} + \lambda_{uv} c_{ij}) x_{ij}^{(u,v)} - \sum_{u,v} \sum_{ij} \mu_{ij}^{(u,v)} z_{ij} - \sum_{u,v} \lambda_{uv} k_{uv} \\
&\text{s.t. } (1), (3) \text{ and } (5) \\
&= \min_{x>0} \sum_{u,v} \sum_{i,j} (\mu_{ij}^{(u,v)} + \lambda_{uv} c_{ij}) x_{ij}^{(u,v)} - \max_{z>0} \sum_{u,v} \sum_{ij} \mu_{ij}^{(u,v)} z_{ij} - \sum_{u,v} \lambda_{uv} k_{uv}. \\
&\qquad \text{s.t. } (1) \text{ and } (5) \qquad\qquad\qquad \text{s.t. } (3) \text{ and } (5)
\end{aligned}$$

This is a dual version of the problem but with only two sets of constraints: preservation of information in the network and degree constraints of nodes. Cost limits and arc-path constraints have been moved to the objective function of this relaxed Lagrangian model, to provide a measure of violation of these constraints. By reformulating the original problem in this form, we can separate it into three independent sub-problems. The second sub-problem

$$-\max_{z>0} \sum_{u,v} \sum_{ij} \mu_{ij}^{(u,v)} z_{ij}, \quad \text{s.t. } (3) \text{ and } (5), \tag{7}$$

can be solved simply by inspection. The first sub-problem

$$\min_{x>0} \sum_{u,v} \sum_{i,j} (\mu_{ij}^{(u,v)} + \lambda_{uv} c_{ij}) x_{ij}^{(u,v)} \quad \text{s.t. } (1) \text{ and } (5), \tag{8}$$

corresponds to n^2 well-defined *shortest path* problems. These can also be solved in a straightforward manner, using existing, computationally competitive algorithms.

One may solve the integer problem using the the *subgradient method* [1]. Once we find a solution to the relaxed Lagrangian model of the LP, we can use *steepest descent* to find a direction and an appropriate step size to go towards a better solution in the next iteration. This is repeated until we converge at a solution for the original ILP.

There are two reasons for pursuing such an iterative algorithm for the relaxed Lagrangian model. First, with a good updating mechanism, this iterative algorithm can converge to a solution within a reasonable amount of time, even for large size networks. Second, this iterative algorithm allows us to develop appropriate distributed algorithms for the problem, which is the ultimate aim of the project.

4 A heuristic approach

While the exact approach is useful as a baseline, it is currently not able to solve problems of the size expected in some real applications, where between 1000 and 10000 users may be expected to be on-line at any one time. We therefore looked at developing a heuristic solution technique able to solve larger problems in a reasonable amount of time.

The heuristic approach is in two phases. In the first phase, a "backbone" is created which ensures all nodes are connected. The backbone observes all the degree constraints. In the second phase, additional arcs are added which try to improve the performance of the network. The intuition here is that if we have an efficient backbone, a large number of the required connections can be made within the requirements. We can then use "spare" connections to improve the performance where maximum time requirements have not been met. We want a backbone that is efficient and which spans (i.e. connects) all nodes. In graph theory, a minimum-weight spanning tree is called a Minimum Spanning Tree. In our case, we have maximum degree constraints, which is known as the Degree-Constrained Minimum Spanning Tree (DCMST).

An interesting question is, how should we weight the MST? There are two options. If we use the maximum time requirements (k_{ij}), then we will tend to link nodes that must be close in the virtual world. This is very useful, and will tend to give the best performance to those who need it most. It may, however, tend to use a lot of long (in the physical world) links. Another approach is to use the actual transmission times (c_{ij}) to weight the MST. This will tend to link nodes that are close together in the physical world. This will mean that each individual link will tend to be short, but many hops may be required to cross the network.

4.1 Creating the backbone

Unfortunately, solving the DCMST is an NP-hard problem [3]. However, some efficient heuristics have been developed [6]. The team developed a method suited to our application. We base this method on Prim's algorithm for the MST (see, for example, [2]). In that algorithm, a tree is built up by adding the lowest weight arc to any currently unconnected node. We can use a variant of this algorithm where we simply place an extra restriction that adding the arc does not break any degree constraints.

We use w_{ij} to denote the weight of a potential arc. As noted in Section 4, we can base the MST on either closeness in the physical network ($w_{ij} = c_{ij}$), or closeness in the virtual world ($w_{ij} = k_{ij}$). The full algorithm is given in Figure 1.

At the end of this procedure, we have a connected backbone. By construction, it observes the degree constraints, and has a low total weight, where weights are derived from either the virtual network or the physical network. Backbones for an example 100-node problem (created using the test problem generator described in Section 5 are given in Figure 2 (based on the physical network) and Figure 3 (based on the virtual network).

4.2 Augmenting the backbone

Once the backbone has been created, extra arcs are added which help to reduce the time to transmit messages between nodes in the virtual network that require small transmission times. Potential arcs (i, j) are considered in order of increasing weight w_{ij}. The weight here is different to the weight used in constructing the backbone. In the simplest version of the

```
• Find the arc $(i_0, j_0)$ which minimises $w_{i_0 j_0}$.
• $B = \{(i_0, j_0)\}$
• while $\exists i \in N$ such that $n_B(i) == 0$
    – Among all potential arcs $(i, j)$ for which
        * $n_B(i) == 0$ and $0 < n_B(j)$ and $n_B(j) < m_j$,
          or
        * $n_B(j) == 0$ and $0 < n_B(i)$ and $n_B(i) < m_i$
    – find the arc $(i', j')$ which minimises $w_{i'j'}$.
    – Add arc $(i', j')$ to $B$.
```

Figure 1: The DCMST algorithm

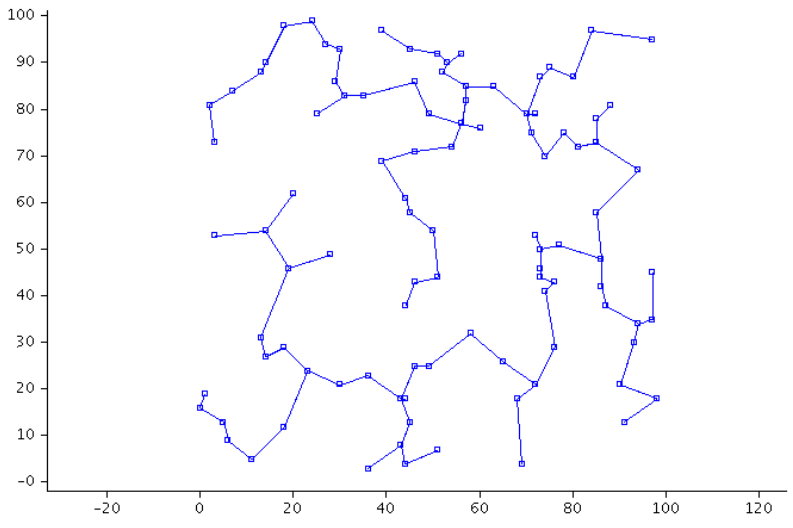

Figure 2: A backbone using transmission time as arc weight

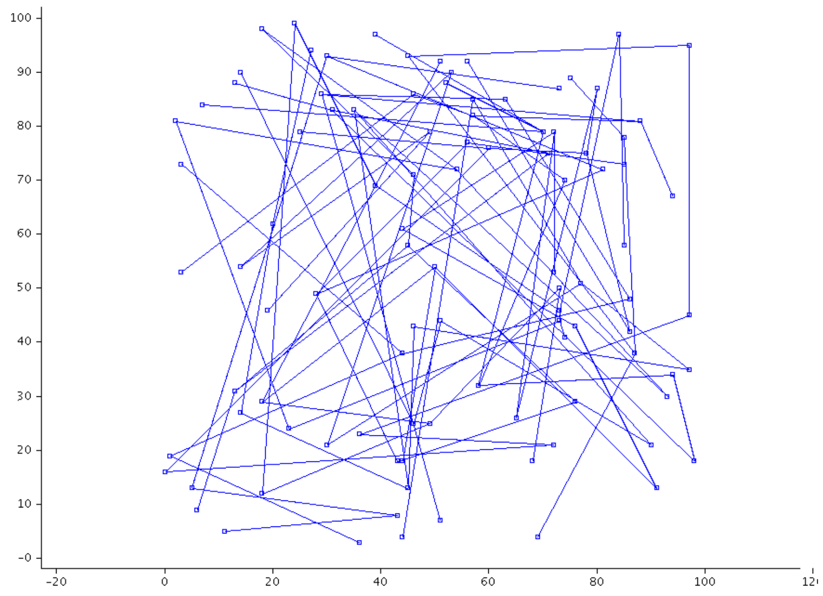

Figure 3: A backbone using maximum allowable time as arc weight

procedure, the weight is exactly $k_{i,j}$, the required transmission time. Variants to this are discussed below.

The arc-set G is initialised to contain all the arcs in B (from the algorithm description in Figure 1). Then, for each arc (i, j) in order of increasing w_{ij}

- if $(i, j) \in G$ they are connected as well as they ever will be. Continue with the next arc.
- If $n_G(i) < m_i$ and $n_G(j) < m_j$ (i.e. neither node has reached its maximum number of neighbours), the arc (i, j) is simply added, and we continue with the next arc.
- Otherwise, we attempt to connect i and j via existing neighbours. We find k_i and k_j such that
  - k_i is i itself, or a neighbour of i in G
  - $n_G(k_i) < m_{k_i}$
  - k_j is j itself, or a neighbour of j in G
  - $n_G(k_j) < m_{k_j}$

  If such one or more such $(k_i,\ k_j)$ can be found, we choose the pair so that $C(k_i, k_j)$ is minimised among all such k_i and k_j, and add (k_i, k_j) to G.

The augmentations considered are shown in Figure 4.

4.3 Algorithm variants

Several variants to the algorithm described in Section 4.2 were explored.

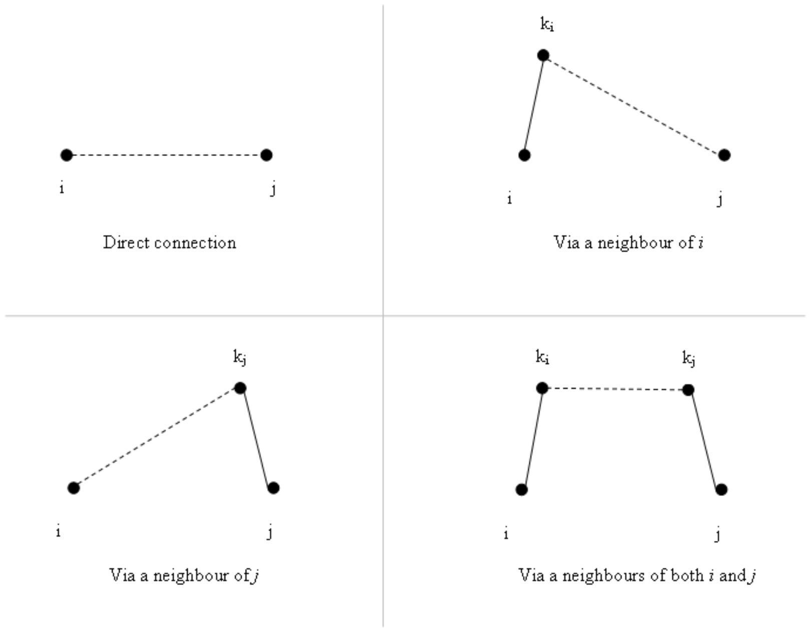

Figure 4: Augmentations considered

4.3.1 Favouring arcs

In the augmentation procedure, we consider potential arcs in order of increasing weight w_{ij}. We can use $w_{ij} = k_{ij}$, the required transmission time, but we also looked at some alternatives.

Favour short arcs Here, we adjust the weight so that amongst arcs with the same k_{ij}, arcs that have low physical transmission times are considered first. The intuition is that we want to use high-speed connections where possible.

$$w_{ij} = k_{ij} + 100C'_{ij}, \quad \text{where} \quad C'_{ij} = \frac{c_{ij}}{\max_{i,j} c_{ij}}.$$

and C_{ij} is a normalised cost

Favour long arcs This is the converse of the above – we adjust the weight so that amongst arcs with the same k_{ij}, arcs that have *high* physical transmission times are considered first. The intuition here is that we want to favour looking at nodes that have low allowable transmission times, but are far apart in the physical world. For this we use

$$w_{ij} = k_{ij} + 100(1 - C'_{ij}),$$

where C'_{ij} is a normalised cost as above.

Favour high-degree nodes We can favour high-degree nodes, so that nodes with the most spare connections are considered first. The intuition is that we want to delay using up scarce connections as long as possible. Here

$$w_{ij} = k_{ij} - m_i - m_j.$$

4.3.2 Randomness

In addition to the favouring methods outlined above, some randomisation may help. So, after the favouring calculation, we set

$$w_{ij} = w_{ij} + rand() * R$$

where $rand()$ is a uniform random number generator given samples in the range $[0, 1)$, and R controls the magnitude of randomness (RandMag).

4.3.3 Path check

Notice that when examining the link between i and j, we do not look at the time required to transmit from i to j through G. Hence we may be using up precious connections to reduce the time for nodes that are already sufficiently "close" in G.

We can add test, after the test for direct connection between i and j:

- If $c_G(i, j) \leq k_{ij}$, continue with the next arc.

The main reason not to look at this feature is the cost of computation. Because the set G is constantly changing, we must calculate $c_G(i, j)$ for each arc considered.

We use an "A*" method (see, for example, [8]), using the direct connection delay c_{ij} as the heuristic estimate function h in that description. This method is able to calculate distances in such a graph very efficiently. Our implementation also stops looking when it can prove that the cost exceeds k_{ij}, also saving time.

4.3.4 MST Improvement

The algorithm that is used to create the degree-constrained minimum spanning tree is a "construct-only" method – it does not try to improve the solution afterwards. It may be possible to find a better DCMST by post-processing the solution.

We looked at an improvement procedure that tries to swap arcs in B for cheaper ones. That is for each $(i, j) \in B$, we try to find a j' such that

- Connecting i to j' is cheaper than connecting via j.
- Replacing (i, j) with (i, j') must also preserve the connectedness of the graph.

All feasible swaps are examined, and the one that decreases the total weight the most is implemented. The procedure is repeated until no more cost-reducing swaps are found.

4.3.5 Double MST

We have mentioned that we can create a backbone based on distance in either physical or virtual networks. But why not both? In this variant, the backbone was created by first using the DCMST based on physical distance. Another DCMST was then created based on virtual distance. Arcs from the second tree were added so long as they did not break the degree constraint for a node. The backbone augmentation procedure (Section 4.2) was then called as usual.

4.3.6 Hubs

One idea developed during MISG week was to try to use a Travelling Salesman Problem tour, rather than a tree, as the basis for the backbone. In the Travelling Salesman Problem (TSP) [7] a salesman must visit each of a set of cities exactly once, then return to their start city, at minimum cost. The advantage of such a tour over a tree is that each node has exactly two neighbours, so the capacity "used up" by the procedure is limited. The idea was to group nodes into clusters with a high-degree node at the centre. These nodes were then connected using a travelling salesman tour.

The way it was implemented for testing was that a search was made for the highest value d such that at least d nodes can be found with at least d spare connections. These d nodes would form the clusters, and be connected to $d-2$ neighbours. We would then use a TSP to connect the d centres. Unfortunately, this procedure cannot guarantee that the graph is connected, so it cannot be used as a backbone. We therefore calculate the hub graph, and add add arcs from the hub to the DCMST backbone.

An example hub graph is given in Figure 5. For the example dataset, there are only 8 centres with (spare) degree 9, but 13 centres with 8 spare connections. Hence those 13 are used as the centres of hubs.

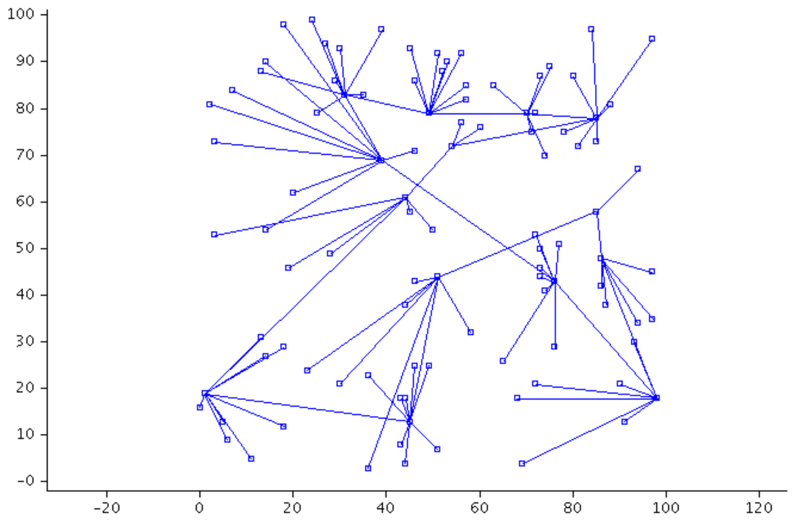

Figure 5: A Hub graph

5 Testing the heuristic

Unfortunately, we were not able to obtain test data from the real world. Instead we developed a test data generator. Essentially, it takes random points in a Euclidean box as nodes on the physical network. The (symmetric) cost to transmit a message is the Euclidean distance, plus a "hop cost" – the delay introduced by passing through a node, regardless of the distance travelled.

hopcost	10
box	100
virtbox	10000
dmean	6.5
dstddev	3
clusbox	200
numinclusave	5
numinclussd	2
minvirtdist	100

Table 2: Paremeters for testing the heuristic

A separate virtual network is made. This world is clustered. Again, distances are Euclidean distances between nodes. The virtual nodes are then randomly assigned to physical nodes. The clustering in the virtual world is achieved by first deciding how many nodes will be in the cluster – the number is drawn from a Normal distribution. A mid-point is selected for the cluster within virtual space, and nodes are distributed uniform-randomly in a box centred on that mid-point. Maximum degrees for a node are selected randomly from a Normal distribution. The generator takes a number of parameters:

size	The number of nodes to generate
box	The size of the (square) box from which physical coordinates are drawn
virtbox	The size of the (square) box from which virtual coordinates are drawn
numinclusave,numinclussd	The parameters of the Normal distribution used to select the number of nodes in a cluster of virtual nodes.
clusbox	The size of the (square) box within which members of a cluster are positioned
hopcost	The extra cost of transmitting through a node
dmean, dstddev	The mean and standard deviation of the Normal distribution from which maximum degree constraints are drawn
minvirtdist	The minimum virtual distance

The k_{ij} are given simply by the Euclidean distance between nodes in the virtual world. Some sanity checks are done

- If the distance is less than the minimum virtual distance given in the parameters, k_{ij} is set to the minimum distance.
- If the virtual distance is less than 1.1 * (the physical distance), the k_{ij} is set to the larger value. Otherwise, the problem is trivially unsolvable.

We generated 100 test problems of size 100, so that we could look at the effect of the algorithm variants.

5.1 Results for size 100 problems

By construction, the degree constraints are always met by the heuristic solution method. In the test problems we generated, not all maximum-time constraints could be met. We therefore treat the deviation from the maximum-time constraints as an objective to be minimised. We looked at two forms of this objective

- Minimise the number of minimum-time constraints violated
- Minimise the sum of minimum-time constraint violations

Variant	Section	Variant values
Backbone Type	4.1 and 4.3.5	**Physical**, Virtual, Both
Favour	4.3.1	**Long**, Short, Max-degree
Randomness (R)	4.3.2	**0**, 25, 50, 100, 200, 500
Path Check	4.3.3	**False**, True
ImproveMST	4.3.4	**False**, True
Double MST	4.3.5	**False**, True
Hubs	4.3.6	**False**, True

Table 3: Algorithm Variants (default in bold)

Objective	Backbone type	10%	Mean	90%	Increase %
Number of Violations	Physical	36	76.3	120	0
	Virtual	26	65.6	102	-15
	Both	18	50.5	84	-2
Sum of Violations	Physical	466	1360	2480	0
	Virtual	548	1880	3000	38
	Both	198	713	1340	-0

Table 4: Backbone type

Table 3 gives the variants examined. The default variant is given in bold. The following sections provide results for each variant. The results for various values the parameter are given, with other values held at the defaults in Table 3. Each table gives

10%: The tenth percentile of the 100 problems – i.e. only 10% of problems had a better value than this. 100 test problems

Mean: The mean value from the 100 problems

90%: The ninetieth percentile for the 100 problems – i.e. only 10% of problems had a worse value than this.

Increase %: The amount by which that variant increases the objective value of the default variant, on average over the 100 problems, expressed as a percentage. So negative numbers mean a better objective.

These are given for both the number of violations, and the sum of time violations. Table 4 shows an interesting result. Using virtual rather than physical distances reduces the number of violations, but then increases the magnitude of those violations. Using both networks does not seem to improve matters. We use backbones based on physical distances in all runs. Table 5 shows that favouring long arcs in augmentation seems to give the best results. Table 6 shows that a small amount of randomness can improve the result – here by more than 10% under either objective. This indicates that it would be worthwhile to make several runs with different seeds and choose the best result. Table 7 shows that checking for an existing connection seems to make a small improvement. Table 8 Improving the MST makes a very small difference – on the whole probably not worth the effort. Table 9 shows that the use of hubs does not seem to improve on the basic DCMST backbone.

Objective	Favour	10%	Mean	90%	Increase %
Number of Violations	Long	36	76.3	120	0
	Short	40	84.6	130	12
	High-Degree	54	96.1	142	32
Sum of Violations	Long	466	1360	2480	0
	Short	564	1530	2550	16
	High-Degree	816	1700	2570	36

Table 5: Favour

Objective	R	10%	Mean	90%	Increase %
Number of Violations	0	36	76.3	120	0
	25	30	67.4	112	-12
	50	30	67.5	110	-12
	100	30	69.8	110	-7
	200	32	75.7	114	0
	500	44	89	132	21
Sum of Violations	0	466	1360	2480	0
	25	348	1140	2140	-18
	50	400	1150	2130	-17
	100	464	1180	2030	-12
	200	442	1290	2080	-3
	500	618	1530	2410	18

Table 6: Randomness

Objective	Do path check?	10%	Mean	90%	Increase %
Number of Violations	False	36	76.3	120	0
	True	32	73.5	122	-3
Sum of Violations	False	466	1360	2480	0
	True	444	1290	2250	-6

Table 7: Path check

Objective	Improve MST?	10%	Mean	90%	Increase %
Number of Violations	False	36	76.3	120	0
	True	32	76.4	126	0
Sum of Violations	False	466	1360	2480	0
	True	450	1370	2480	-1

Table 8: Improve MST

Objective	Use hubs?	10%	Mean	90%	Increase %
Number of Violations	False	36	76.3	120	0
	True	44	82.5	130	11
Sum of Violations	False	466	1360	2480	0
	True	658	1500	2560	15

Table 9: Hubs

5.2 Results for larger problems

Using the insights gathered from the smaller problems, we ran tests on some problems with 1000 nodes. These are closer to the size of problems that will be encountered in practice. The algorithm variant used was

Backbone Type	Physical
Favour	Long
Randomness (R)	0
Path Check	True
ImproveMST	False
Hubs	False

We note that using some randomness may improve results, but decided to use no randomness to give a more stable baseline. The results are

	10%	Mean	90%
Number of Violations	1120	1310	1500
Sum of Violations	18200	27800	34200

Thus, of the 10^6 possible connections, less than 0.2% fail to meet the performance requirement. However, execution time for this version was rather long[1]. Average execution time was around 130 seconds. However, if we do not use the path-check option, solution time decreases significantly. The following results were achieved with Path Check set to false:

	10%	Mean	90%
Number of Violations	1090	1300	1470
Sum of Violations	18300	27800	33800

These results are essentially the same. However the execution time was less than 3 seconds on average, split equally between spanning tree calculation, and heuristic augmentation. Interestingly, using both physical and virtual spanning trees (that is, Backbone Type "Both") gave a significant improvement in the larger size problems.

	10%	Mean	90%
Number of Violations	642	929	1110
Sum of Violations	10700	19900	25300

[1]Our testing machine was a desktop Linux box (dual core, Intel Pentium-4 3600 MHz processors, 2GB cache)

This was also very slow when using the path-check option – execution times around 2.5 minutes. Again, simply skipping the path-check speeded up execution without notably decreasing quality:

	10%	Mean	90%
Number of Violations	662	933	1190
Sum of Violations	12100	20100	24600

Execution time without path check was around 4.5 seconds. We would therefore recommend this as the variant with best prospects of success.

5.3 Results for small problems

In order to see how far from optimal the heuristic solutions are, we ran the heuristic procedure on the small problems tested in Section 3.2. We generated five solutions to each problem, and selected the best. Execution times in all cases were negligible ($<$ 0.01 seconds). The results are presented for the standard combination of parameters (set 1) and a set which gives improved results (set 2). So, even though we know that feasible solutions exist, only two of these solutions could be found using the standard parameter values (set 1). Exploring other parameters values, we found that set 2 gave better results. This shows that while the heuristic is able to produce some good solutions, it fails to find good solutions when we know they exist. Unfortunately, we were not able to compare the exact and heuristic solutions, to see if there were systematic features that could be used to improve the heuristic approach. We must leave this for future exploration.

Backbone Type	Physical
Favour	Long
Randomness (R)	0
Path Check	False
ImproveMST	False
Hubs	False

Backbone Type	Physical
Favour	Long
Randomness (R)	200
Path Check	True
ImproveMST	True
Hubs	False

Table 10: Parameter values, set 1 (left) and set 2 (right)

Prob	Number of Violations	Sum of Violations
#1	2	2
#2	6	126
#3	2	16
#4	2	2
#5	2	54
#6	2	6
#7	0	0
#8	12	142
#9	0	0
#10	10	198

Prob	Number of Violations	Sum of Violations
#1	2	6
#2	2	30
#3	0	0
#4	2	2
#5	2	58
#6	2	6
#7	0	0
#8	12	70
#9	0	0
#10	8	104

Table 11: Results for small problems: parameter set 1 (left) and set 2 (right)

6 Conclusions

We have formulated an exact ILP model for the problem of communicating on a virtual network. While this ILP model was successful in solving small problems, it is not recommended to handle larger instances, due to the fact that the number of variables in the model grows exponentially as the graph size grows. However, this ILP model can provide a benchmark for heuristic algorithms developed for this problem. We have also described a heuristic approach, and explored several variants of the algorithm. We found a solution that seems to perform well with reasonable computation time. The heuristic is able to find solutions that respect the degree constraints, but show a small number of violations of the desired time constraints. Tests on small problems show that heuristic is not always able to find feasible solutions, even though the exact method has shown they exist. It would be interesting in the future to look at whether insights gained by looking at exact solutions can be used to improve the heuristic. We look forward to obtaining some real-world data to see if the heuristic algorithm (or one of its variants) gives acceptable results.

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the ARC through the ICT Centre of Excellence program. The third author is supported by the ARC Discovery Grant DP0984470. The authors would like to thank the team of contributors Pam Davy, Gregory Doherty and Jacqui Ramagge, and the AMSI Summer School students Pierre Braun, James Holland, Ge Li, Michael Sun, Robert Tang and Fan Wu. The authors are also grateful to Michael Haythorpe for his assistance.

References

[1] Bertsekas, D.P. (1999) *Nonlinear Programming*, Athena Scientific, Cambridge, MA.

[2] Cormen, T.H., Leiserson, C.E. & Rivest, R.L. (1989) *Introduction to Algorithms*, The MIT Electrical Engineering and Computer Science Series, The MIT Press, Cambridge Massachusetts.

[3] Garey, M.R. & Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco.

[4] Jiang, X., Safaei, F. & Boustead, P. (2005) Latency and scalability: A survey of issues and techinques for supporting networked games, In *IEEE 7th Malaysia International Conference on Communication and the 13th IEEE International Conference on Networks.*

[5] Jiang, X., Zafaei, F. & Boustead, P. (2007) An approach to achieve scalability through a structured peer-to-peer network for massively multiplayer online role playing games, *Computer Communication*, **30**, 3075-3084.

[6] Krishnamoorthy, M., Ernst, A.T. & Sharaiha Y.M. (2001) Comparison of algorithms for the degree constrained minimum spanning tree, *Journal of Heuristics*, **7** (6), 587-611.

[7] Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G. & Shmoys, D.B. (1985) *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley and Sons, Chichester.

[8] Russell, S. & Norvig, P. (1995) *Artificial Intelligence: A Modern Approach*, Series in Artificial Intelligence, Prentice Hall, Englewood Cliffs, New Jersey.