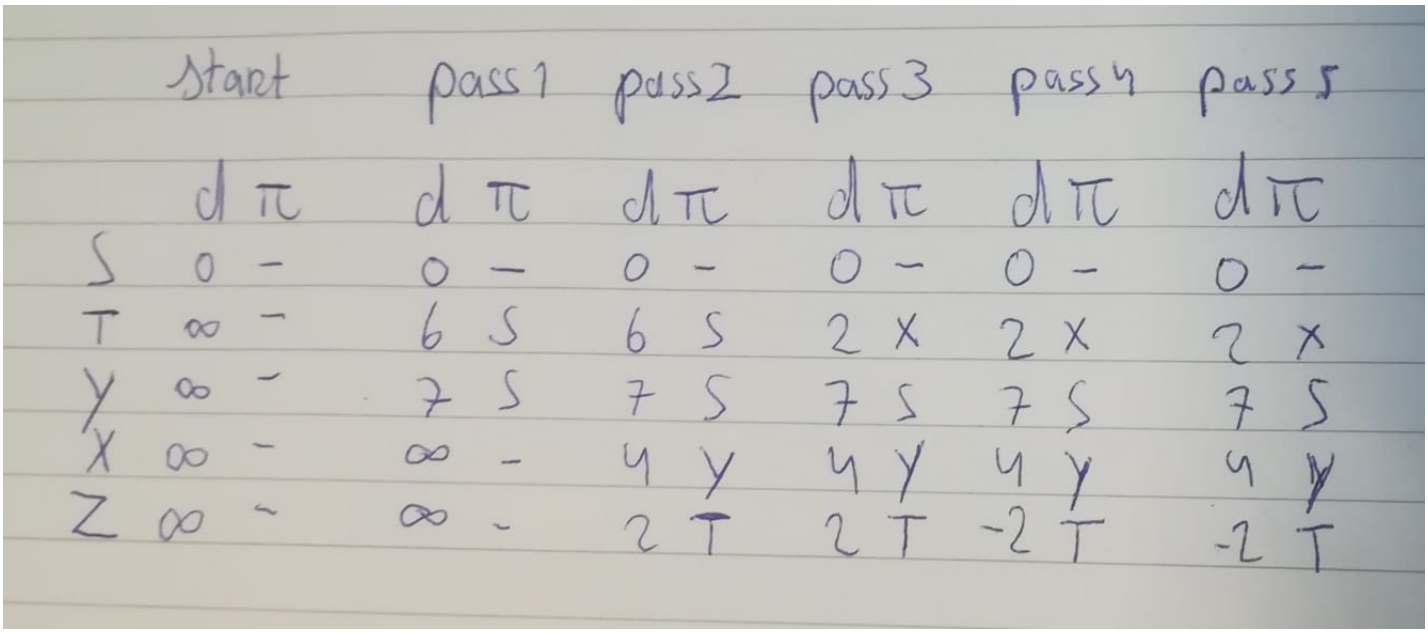
Problem session 5

Dibran Dokter 1047390 & Marnix Lukasse 1047400

October 4, 2019

5

5.1



5.2

We found the sentence "an amount S in currency A costs S · c(A, B) in currency B" rather confusing. We assumed it was meant that in the given graph the following sentence is true, "an amount S in currency A **gives you** S · c(A, B) in currency B".

1) The exchange rate from a_1 to a_k is equal to $c(a_1, a_2) \cdot c(a_2, a_3) \cdot ... \cdot c(a_k - 1, a_k)$.

2) If there is a cycle in the graph and the exchange rate of the cycle is > 1.(The exchange rate can be determined as shown in the previous answer).

3) We compare the exchange rate from s_1 to that of s_2 times the exchange rate from (C,B). Whichever one is higher is the better path. (Compare (A,B) with (A,C) · (C,B)

4) We apply bellman-ford but instead of adding the costs together we multiply the exchange rates and take the highest one. For example where the normal bellman-ford would compare 1.1 with 1.1+1.2 and then take the lowest one. This modified version compares 1.1 with 1.1 · 1.2 and then takes the highest value.

Since bellman-ford already has been proven to find the shortest path (with the lowest cost), it will also be able to find the path with the highest exchange rate if we use the new comparison as described above.

This will result in finding the highest exchange factor from a vertex A to B. The highest exchange factor is the same as the highest exchange rate, so we can find all exchange rates this way.

5.3

To find this we run a slightly altered version of BFS from vertex s where s is the root in our given SPT. The thing we alter in BFS, is that a node can have multiple predecessors. Normally BFS only has one predecessor, because if we find a different route where the cost is equal to the current cost, we don't update the predecessor as there is no point in doing so. In this case we'd like to have a list of all the predecessors that are part of the lowest cost route to the vertex, there could be multiple 'shortest' paths.

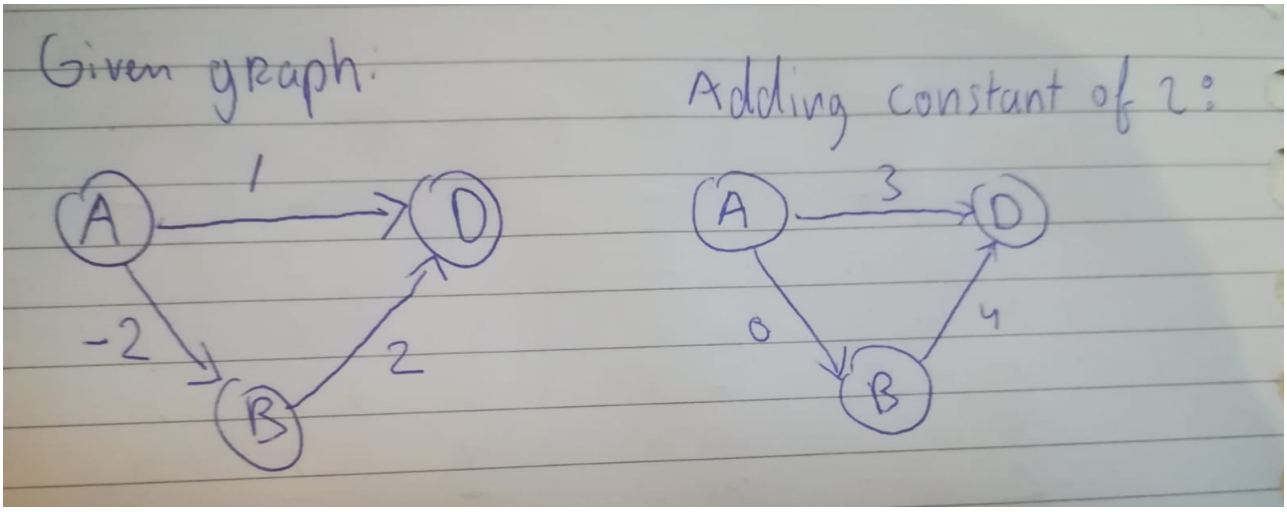
After doing this version of BFS (which is linear), we have a look at our given SPT, and for each edge (starting from all leaves going upwards) we do the following check: Lets say we have vertex B in SPT with predecessor A, then we have a look at BFS and check whether A exists in the list of predecessors of B. If this is case, then that edge in the SPT does not conflict with our BFS results. We do this for all edges in the SPT, if there are no conflicts in any of the edges we can say that given SPT is indeed a correct SPT for the given graph.

Since BFS = $\mathcal{O}(|V| + |E|)$ this algorithm is linear.

5.4

This algorithm does not work in all cases. In certain cases the shortest path in the original graph will not be the shortest path in the graph where we added a constant to all the edges.

For example this graph:



In this graph the shortest path in the original graph is A->B->D. When we add a constant (in this case 2) the shortest path becomes A->D. This is incorrect. This is because the path can have a larger amount of edges. Thus we add more to the bottom path than the upper path.

5.5

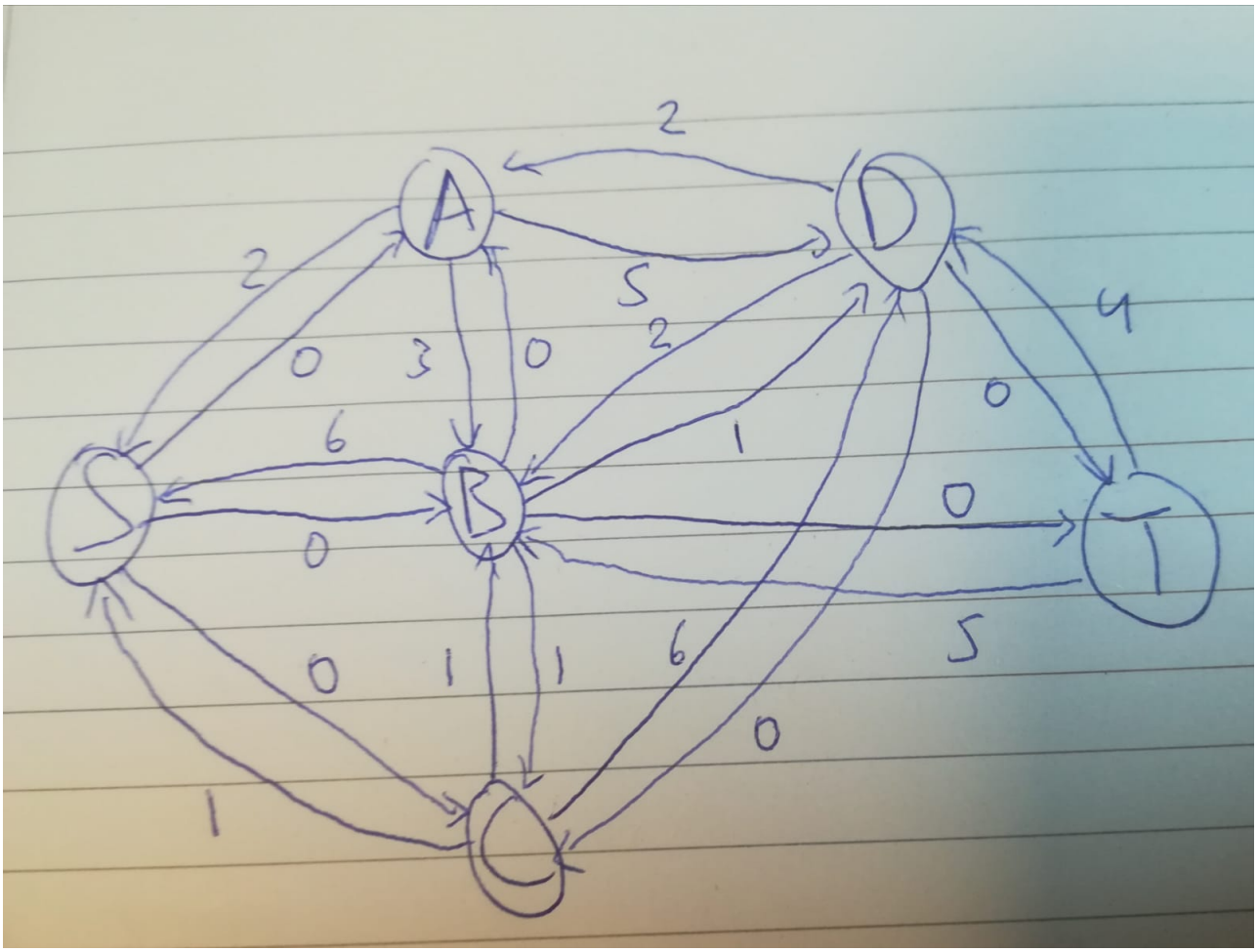
Because our residual graphs for the steps were rather

5.6

1) This problem corresponds to a maximum flow problem. Thus we can use algorithms that find the maximum flow through a graph. In this case we can use Ford-Faulkerson to solve this problem.

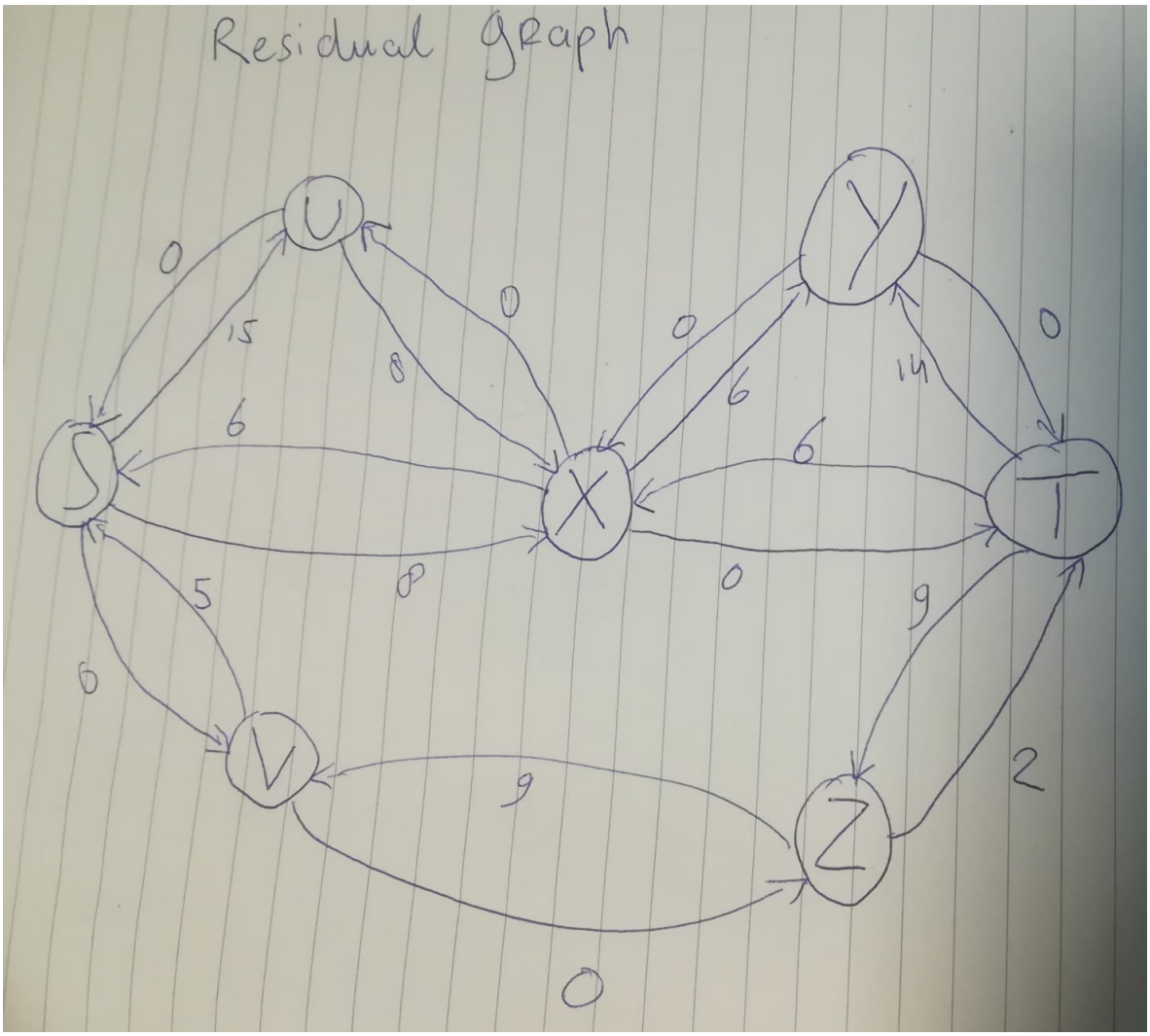
2) We were unsure if we needed to find only one path from S to T and optimize that or that we needed to find the maximum flow via more than one path. In the end we assumed that we need to find the maximal flow.

To get the maximal flow we ran the Ford-Faulkerson algorithm and came to the following residual graph:



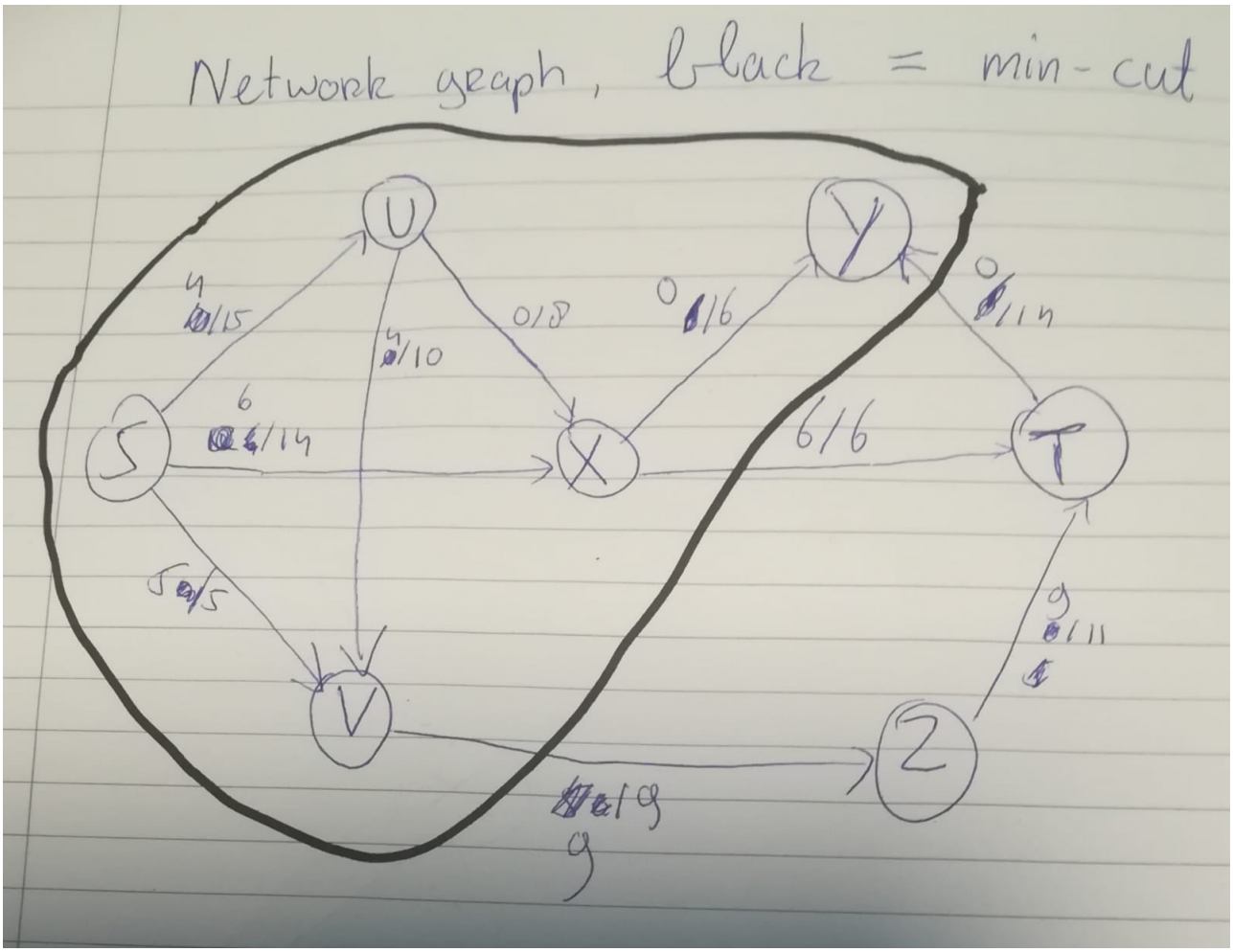
3) From the above residual graph we can see that the maximal bit rate is 9. The route that enables this is also seen in the residual graph. messy we have redrawn the final residual graph so that it was actually readable.

Residual graph:



As we can see in the residual graph the maximal flow is 15. Caused by the bottlenecks from V->Z of 9 and the bottleneck from X->T of 6.

Min-cut:

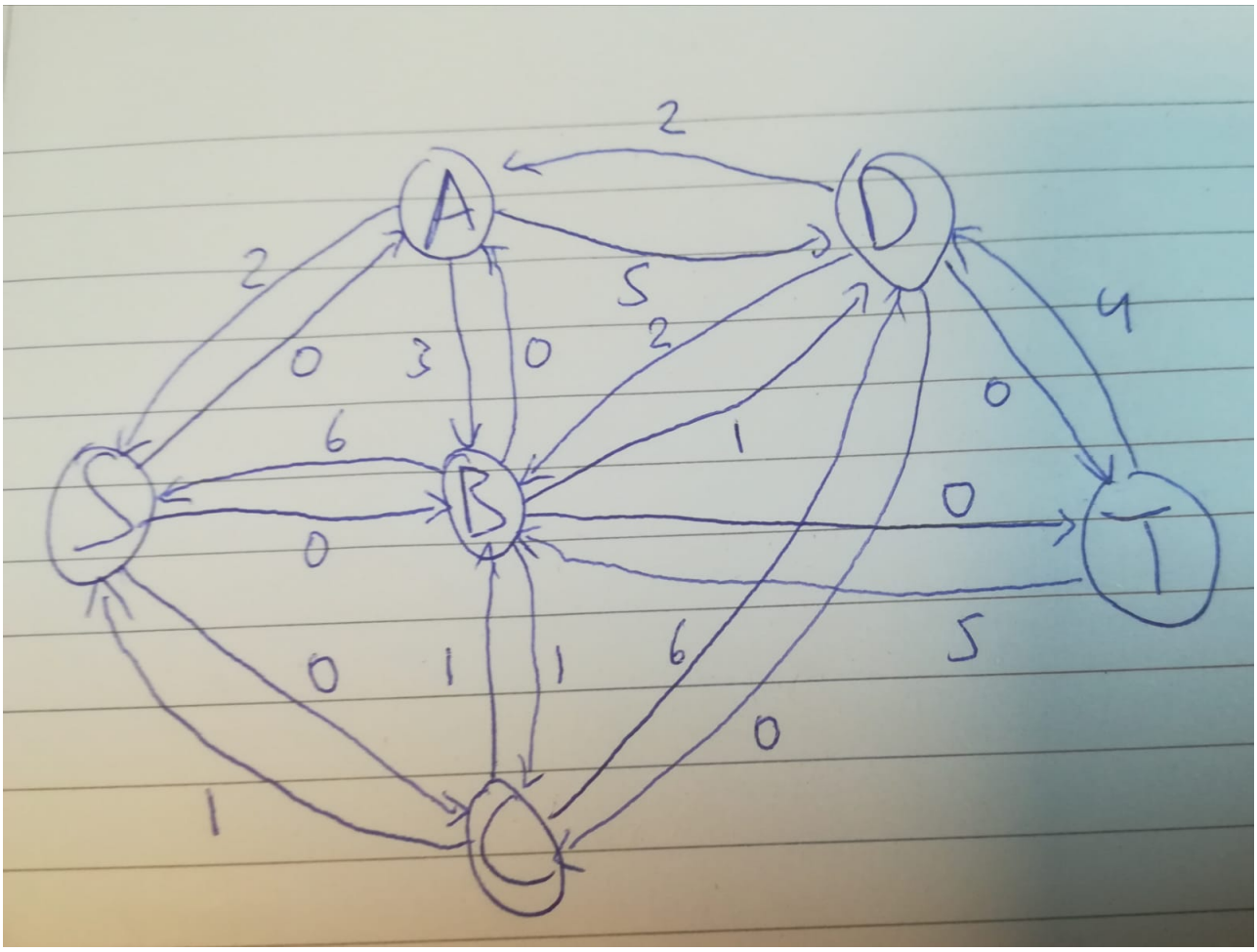


5.6

1) It wasn't entirely clear to us whether we should find the single route with the most throughput, or if we are allowed to use multiple routes simultaneously. We assumed the latter was the case.

This problem corresponds to a maximum flow problem. Thus we can use algorithms that find the maximum flow through a graph. In this case we can use Ford-Faulkerson to solve this problem.

2) To get the maximal flow we ran the Ford-Faulkerson algorithm and came to the following final residual graph:



3) From the above residual graph we can see that the maximal bit rate is 9 Mbit/s, using 4 different routes simultaneously:
- 2 Mbit/s S -> A -> D -> T
- 5 Mbit/s S -> B -> T
- 1 Mbit/S S -> B -> D -> T
- 1 Mbit/s S -> C -> B -> D -> T