

# Modeling and Simulation of the Performance of Distributed Video Services

*Networks 2008, Budapest  
2nd October, 2008*

**Péter Kárpáti**, Norwegian University of Science and  
Technology (NTNU)

**Tibor Szkaliczki**, Computer and Automation Research  
Institute of the Hungarian Academy of Sciences, eLearning  
Department

**László Böszörményi**, University Klagenfurt, Department  
of Information Technology

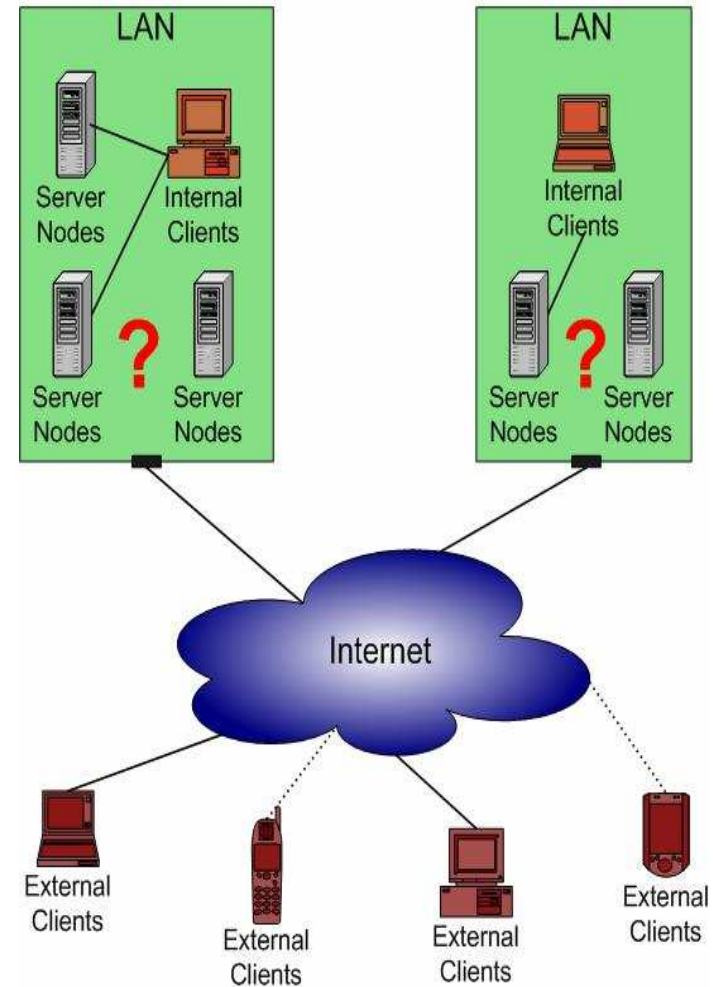


# Overview

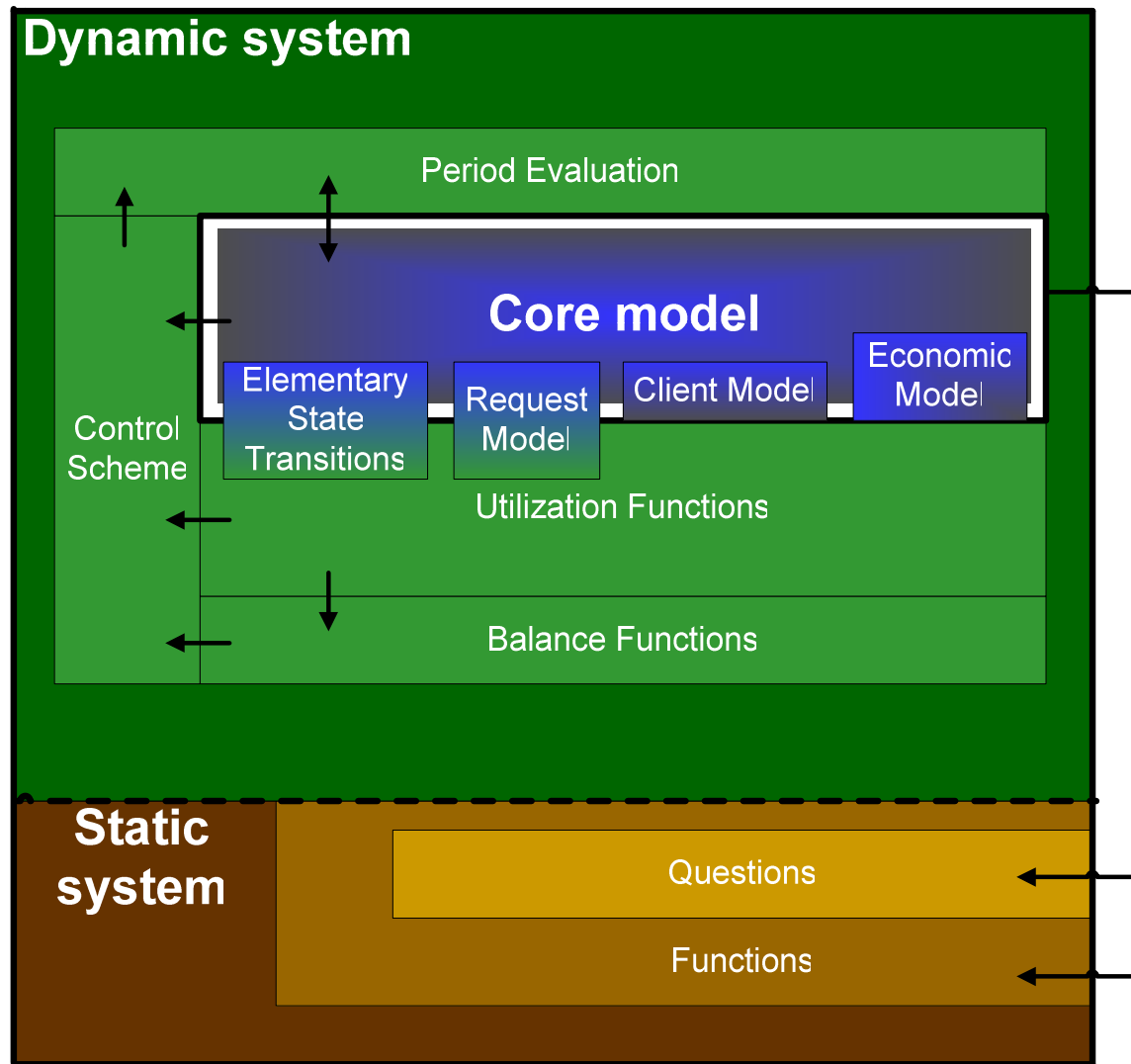
1. Motivation
2. Formal model
3. Simulation model
4. Performance study based on the simulation model
5. Conclusions

# I. Motivation

- Scaling VoD servers
  - How much node and network resource is needed to serve the expected clients?
  - Static servers are usually designed to the maximum expected client load thus probably oversized
  - Performance study of self-organizing behaviours



## 2. Formal model



## State space

$$\mathbf{SSys} = \mathbf{Subnets} \times \mathbf{DirectConns}$$

$$\mathbf{Subnets} = P(\mathbf{Subnet})$$

$$\mathbf{Subnet} = \mathbf{SId} \times \mathbf{Nodes} \times \mathbf{InsideNetThp} \times \mathbf{IngoingNetThp} \times \mathbf{OutgoingNetThp}$$

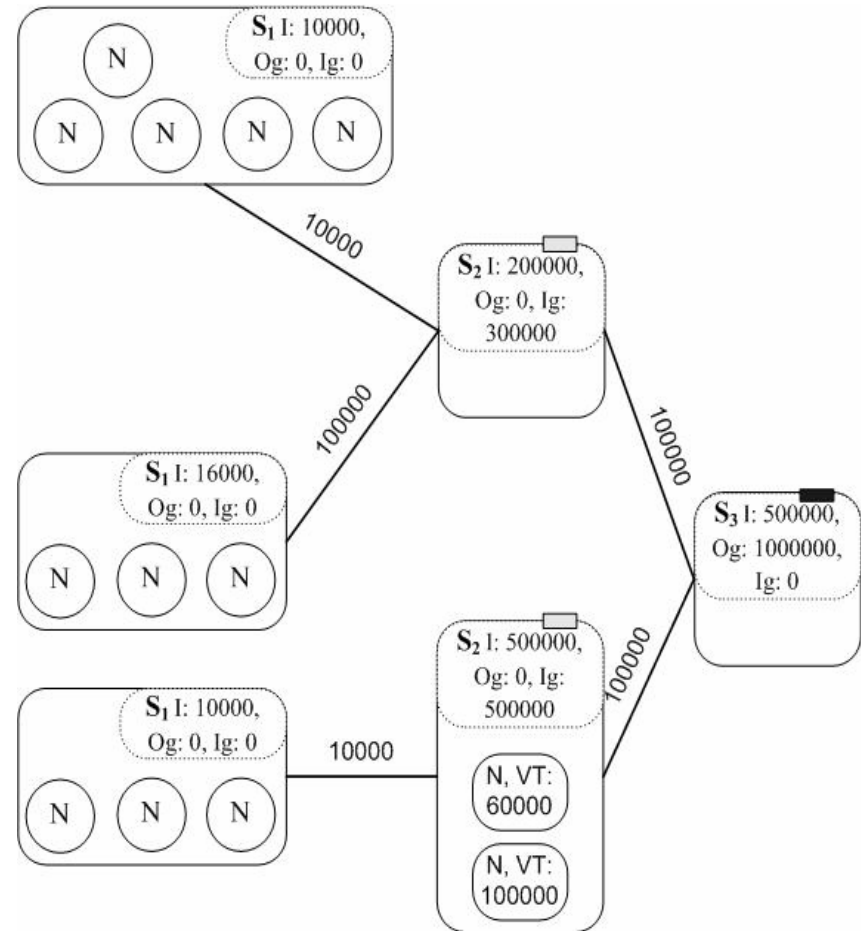
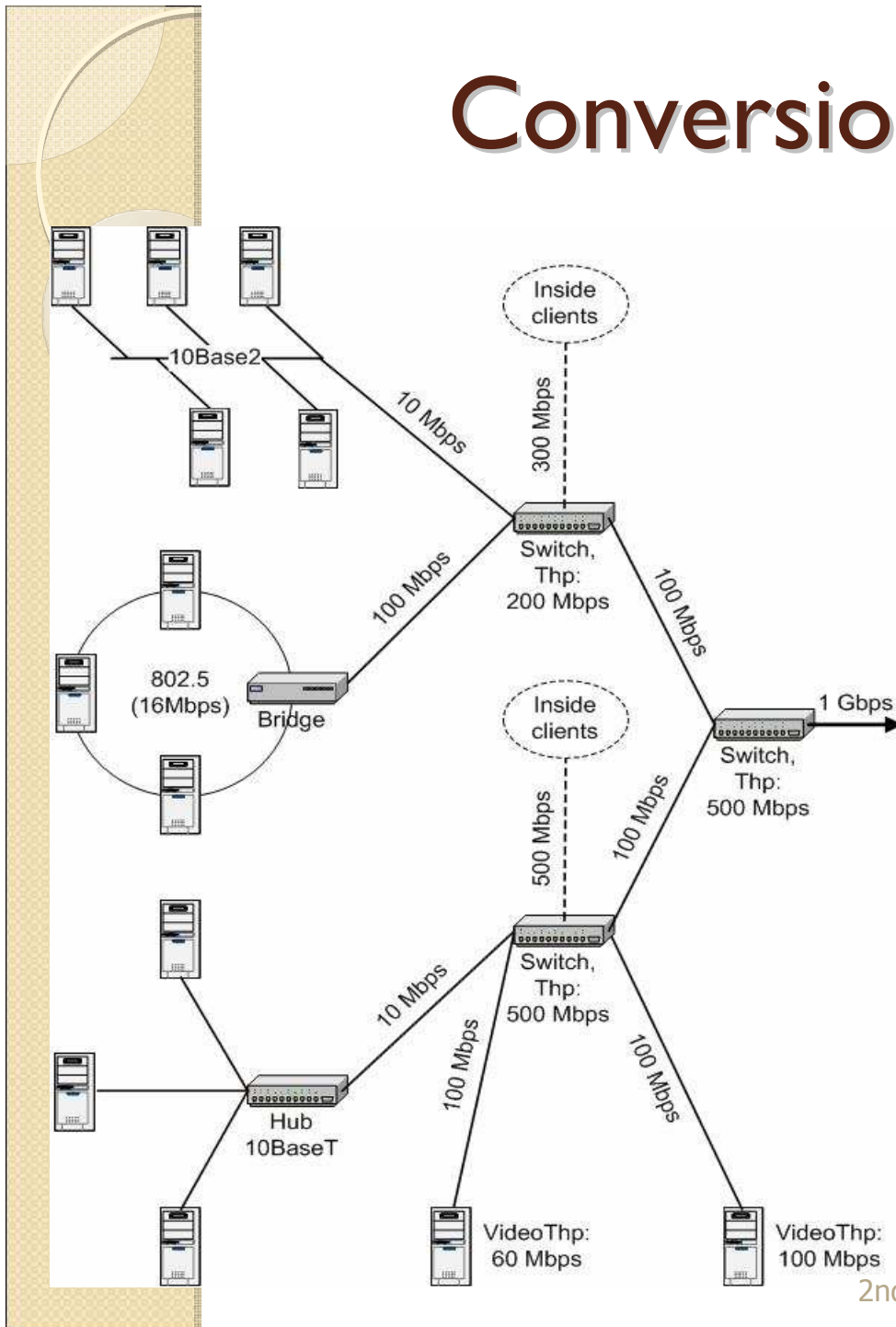
$$\mathbf{Nodes} = P(\mathbf{Node})$$

$$\mathbf{Node} = \mathbf{NId} \times \mathbf{Sto} \times \mathbf{VideoThp} \times \mathbf{StoRAccThp}$$

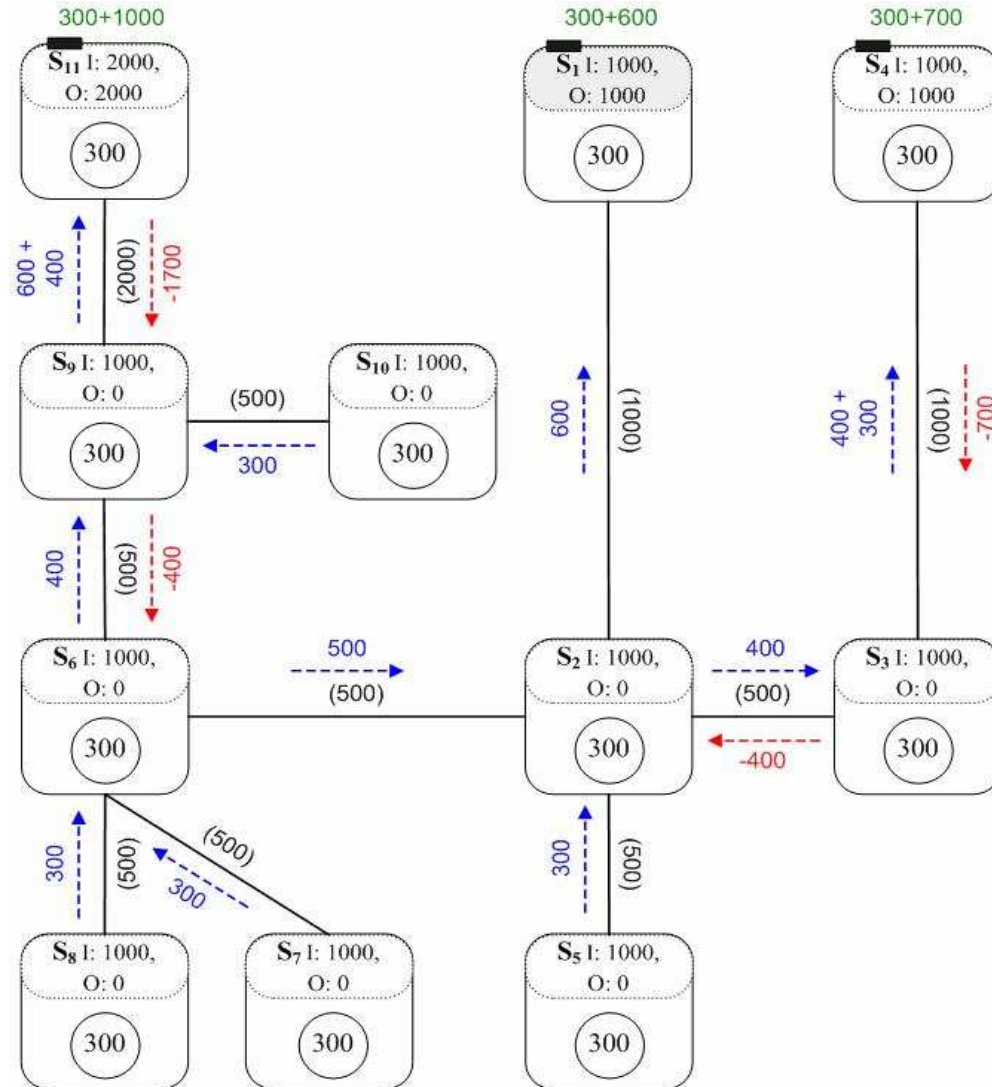
$$\mathbf{DirectConns} = P(\mathbf{DirectConn})$$

$$\mathbf{DirectConn} = \mathbf{SAId} \times \mathbf{SBId} \times \mathbf{NetThp}$$

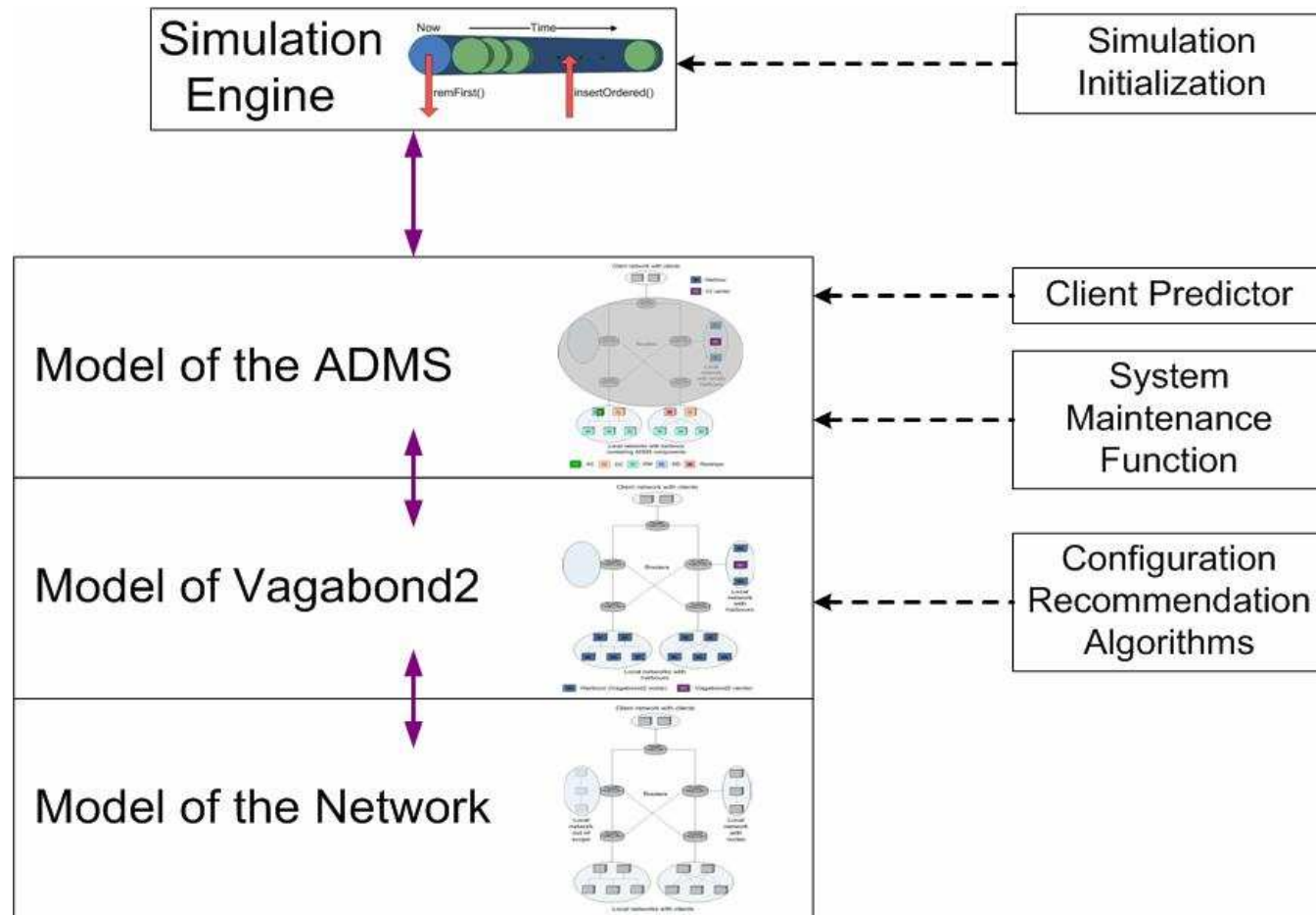
# Conversion example



# Potential traffic directed to clients



# 3. Simulation environment



Discrete, event-based simulation

# Modelling the network

- Network entities
  - Router
  - Local network
    - Nodes (CPU, memory, disk, disk access): computers for the VoD server
    - Connection structure of nodes unknown
    - Throughput can be derived from a real LAN by the *swallowedTraffic()* algorithm
  - Client network
    - Nodes (connection bandwidth): client terminals
- Connections
  - Characterized only by bandwidth currently



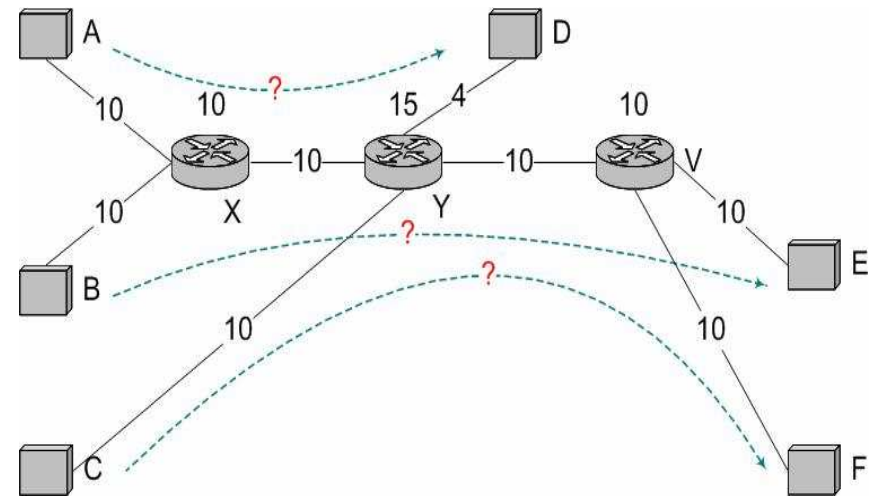
# Simplifying reality

## Assumptions

- no failures appear in the network
- all the resources can be fully used with their specified values
- the connections are symmetric, asynchronous and bidirectional
- the connections and network entities do not introduce any delay for a transmission
- there is no other traffic
- the routes do not change during a given transaction

# Simulating the traffic

- Two types: streaming and replication
- Problems with simulating replication
  1. Its end should be scheduled in the simulation  
→ ReplicationReservationFunc()
  2. More simultaneous replication → how to provide TCP-like fair traffic distribution



# Traffic distribution algorithm

- **Additional assumptions**
  - If  $N (>0)$  data transmissions use a network element, each stream gets the  $N$ -th part of its throughput
  - Each data transmission is assigned to a route
- **Base idea**
  - The value of the smallest bottleneck in the given transmission routes will not change because
    - the routes are fixed and thus
    - the routes crossing this bottleneck can send their traffic only at the rate according to this bottleneck throughput

## 4. Performance study based on the simulation model

- Static vs. semi-dynamic vs. dynamic behaviour strategies for VoD servers
- Network topology
  - generated by GT-ITM package, Transit-Stub model
  - 39 routers, 13 LANs, 4 owned nodes per LAN
- Load
  - Generated by MediSyn, synthetic
  - ~113 000 requests, 2000 videos in 10 genres, duration of a video bw. 3 min and 2.5 hours

## Study parameters (cont.)

- One run covers 30 weeks
- One bundle of the runs
  - 3 types of systems
  - 7 resource distribution
- 8 bundles with identical statistical characteristics =>  
168 runs

	few, single	few, double	plenty
Storage (GB)	40	80	600
Bandwidth (stub- gateway-transit) (Mbps)	10-20- 40	20-40- 80	500- 1000- 2000
Node res. (Mbps)	6	12	300

# Performance metrics

- Average streaming path length
- Average transmission path length
- Network load per served requests

Network load =

$$\sum_{a \in \text{StreamingAndReplicationActions}} (a.duration * a.usedBandwidth * a.numberOfHops)$$

- Maximum streaming bandwidth

## Summary of results

- Dynamic behaviour strategy is better than the other two in the presence of bottleneck resources
  - Served more than 77 000 requests out of almost 113 000 while the others less than 48 000 requests (all in average)
  - Placed the videos 2-4 hops nearer to the clients
  - Produced usually the lowest relative network load
- Semi-dynamic strategy did not match expectations
- Static strategy was most favourable in the case when “plenty” resources were available

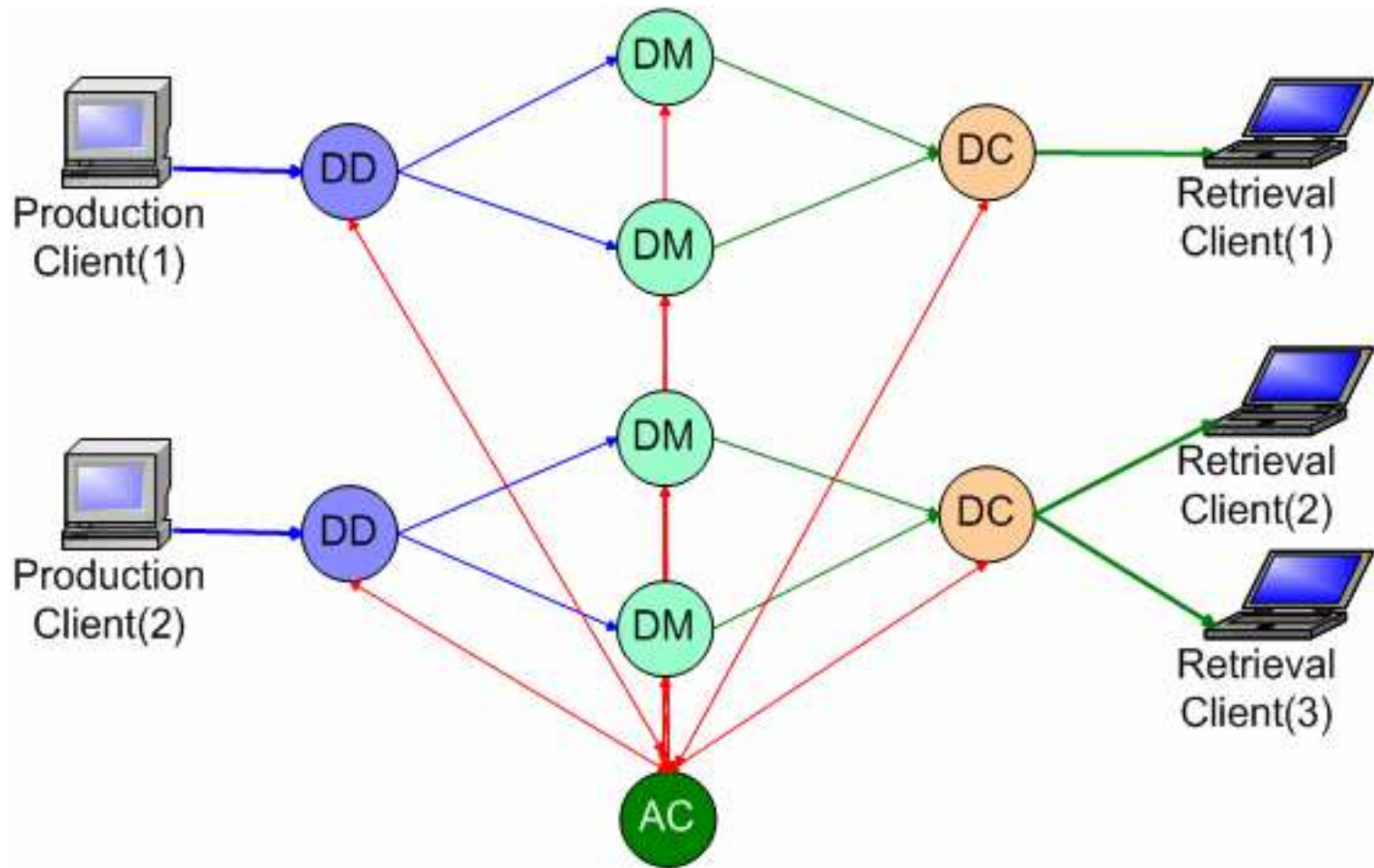
## 5. Summary

- Performance depending on the underlying network detached from the server's potential service performance by the models => optimization of the two kinds can be done separately
- The two models are general enough to be useful for other network-based applications
- It should be investigated how the models perform when applied for real systems



**Thank you for your attention!**  
**Questions?**

# Self-organising VoD servers - ADMS



# Functions for different cases

Type	Description	Model
0.	Resources fully used, nothing happens	Not in the model
1.	The worst resource affects the whole system.	In the topology-independent case
2.	The worst resources affect the independent parts of the system (DCS's / LANs) separately.	In the topology-independent case
3.	The worst resources affect the parts of the system depending on certain assumptions.	In the topology-dependent case

"Type-1 worst case"  $\leq$  "Type-2 (TI) worst case"  $\leq$  "Type-3 (TD) worst case"  $\leq$  "TD best case"  $\leq$  "TI best case"

# Results

CI-s by 90-quantile	Avg. streaming path length	Avg. transm. path length	Network load per served req.	Max. streaming bandwidth
S-SD: f.s. bw.	-0.494, -0.036	0.419, 2.02	3239, 9910	0.156, 2.019
S-D: f.s. bw.	-0.049, 0.386	3.328, 4.97	12796, 19843	-26.142, -24.083
S-SD: f. d. bw.	-0.468, -0.049	0.26, 1.916	6465, 17320	0.427, 4.698
S-D: f. d. bw.	-0.054, 0.397	3.242, 4.83	27767, 37591	-49.627, -43.223
S-SD: f.s. node	0.0006, 0.427	-1.485, -0.163	-14791, -6941	-1.965, -0.335
S-D: f.s. node	0.026, 0.165	-0.822, -0.136	-9729, 10276	-55.647, -46.403
S-SD: f. d. nod	0.039, 0.36	-0.726, 0.098	-9707, -590	-3.285, 0.66
S-D: f. d. node	0.046, 0.16	-0.674, -0.028	-35276, -19671	-128.75, -99.849
S-SD: f.s. sto.	-0.077, 0.064	-0.126, 0.11	-2185, 1206	-2e-014, 8e-014
S-D: f.s. sto.	-0.604, -0.351	1.193, 1.474	4383, 18139	67.171, 126.953
S-SD: f. d. sto.	-0.117, 0.062	-0.149, 0.115	-1871, 2066	0, 0
S-D: f. d. sto.	-0.697, -0.386	1.697, 1.972	17904, 28540	-4.908, 53.458
S-SD: plenty	-0.09, 0.148	-0.004, 0.356	87, 6028	0, 0
S-D: plenty	1.112, -0.488	2.458, 3.376	36574, 53005	-46.67, 35.844

# Traffic distribution algorithm

## Additional assumptions

- If  $N (>0)$  data transmissions use a network element, each stream gets the  $N$ -th part of its throughput there is no other traffic
- Each data transmission is assigned to a route

## Input

- vector of the routes: *Route*
- table of the available throughputs: *ThpTable*

## Output:

- required throughputs

## Algorithm

1. Create the table *FreqTable*
2. Create the vector *AvgThpRoutes*
3. Find the route(s) with the smallest bottleneck throughputs:  
*SmBnThpRoutes* in *AvgThpRoutes*.
4. Assign the computed smallest bottleneck throughputs to the routes crossing this(these) point(s). These routes are fixed.
5. If no more routes then **finish**.
6. Update the *ThpTable* and *FreqTable* in accordance with step 4.
7. Repeat the calculation for the remaining routes from 2.