TuneInNet – Flooding on the Internet Backbone

Diplomarbeit

Sebastian Dill (dill@inf.fu-berlin.de)

Betreuer: Prof. Dr.-Ing. Jochen Schiller
Georg Wittenburg, M.Sc.

Overview

Freie Universität Berlin

- **Motivation: Clean Slate Internet**
- **Concept & Design of TuneInNet**
- **Graph Theoretical Considerations**
- **Prototype Implementation**
- **Evaluation Setup**
- **Evaluation**
- **Conclusion**

State of the Network

- **Current Internet has well-known problems**
  - Scalability (BGP routing table growth, address shortage)
  - New application types/ requirements (QoS, Mobility, Multicast)
- **Initial conception of the Internet did not anticipate growth and new uses, solutions are often suboptimal**

↓

- **Revisability of existing infrastructure may reach its limit**

↓

- ***Clean slate* research investigates how an entirely new infrastructure would be build from scratch with today's knowledge**

Clean Slate Research

- **Disregards practical and economic constraints**
- **Can challenge the fundamentals of the current technology**
- **Can explicitly persue new paradigms in a radical manner**
- **Most Clean Slate approaches propose more complex architectures**
- **TuneInNet proposes to radically eliminate the remaining complexity from the network**

Background

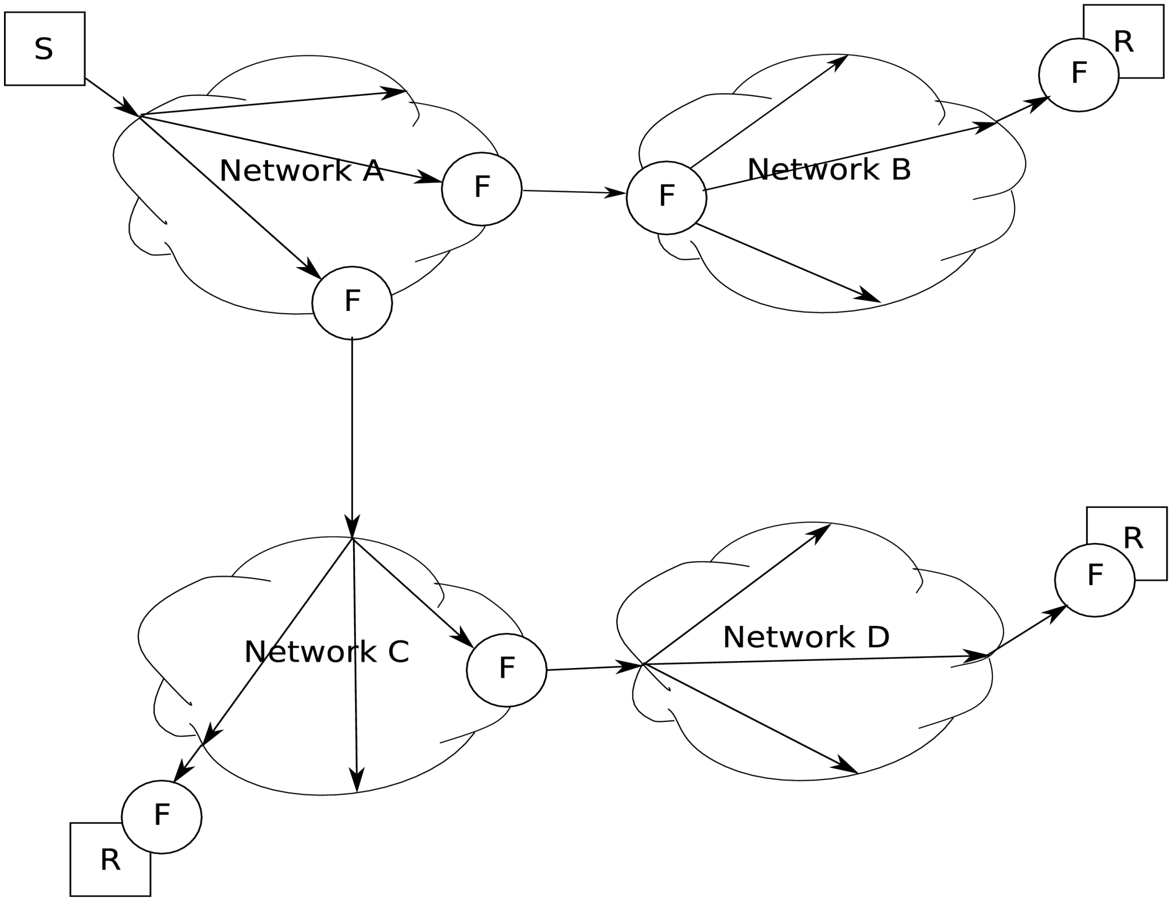
Freie Universität Berlin

- **Success of Internet partly because “intelligence is end to end rather than hidden in the network.” (RFC1958)**
- **“raw connectivity” view: interpret current problems as result of remaining complexity**
- **Radio is example of raw connectivity: sender blindly sends, receiver filters out (tunes in to) interesting data**
- **TuneInNet attempts to transfer this to the Internet, aiming at providing essentially OSI layer 2 functionality**
- **This thesis evaluates TuneInNet in its most radical form**

TuneInNet – Basic Concept

Freie Universität Berlin

- **Sender Individualizes packet somehow and sends it into home AS**
- **Every network floods packet internally so it reaches all exits to peering AS**
- **Networks can apply filters at entry and exit points**
- **Receivers filter out data relevant to them**
- **Only method to remove circling packets is a TTL field**



Advantages and Disadvantages

Freie Universität Berlin

Advantages:

- **Very simple, purely optical network devices**
- **Possibly more suitable for streaming applications**
- **Mobility Problem is solved**

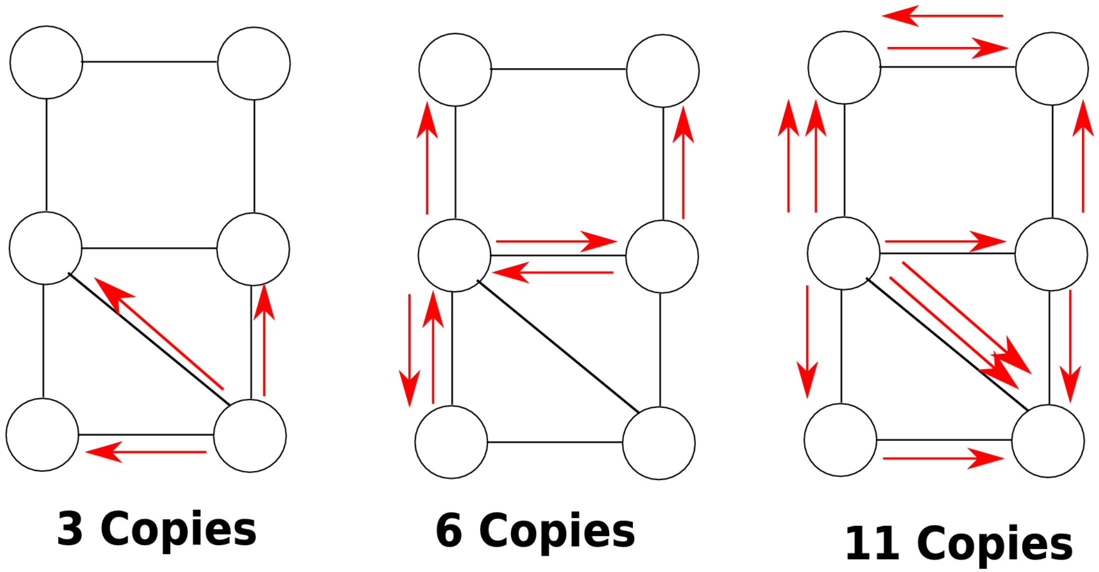
Disadvantges:

- **Traffic Overhead**
- **Security Concerns**
- **Filters may re-introduce complexity**

Would a decreased cost in network devices offset the higher bandwidth cost caused by the traffic overhead?

Thesis gives answer for radical version of TuneInNet

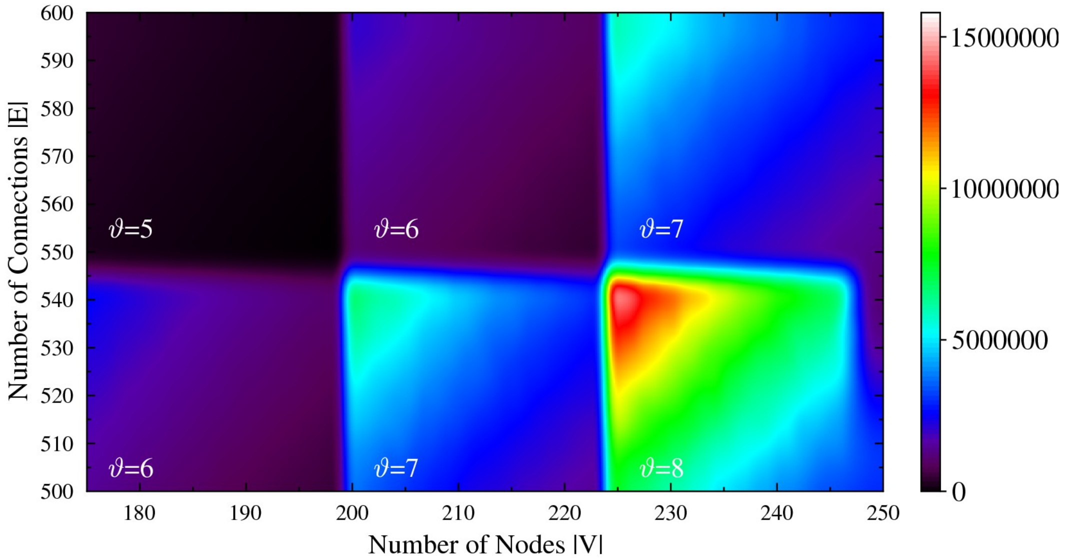
Total Traffic Modifier ϕ



- Given a start node, it is possible to calculate how often a a packet will be sent along any network link
- The variable ϕ(T) is the average over that value for all nodes in a topology T
- It is a general indicator of how much traffic to expect in T

Total Traffic Modifier ϕ

Freie Universität Berlin

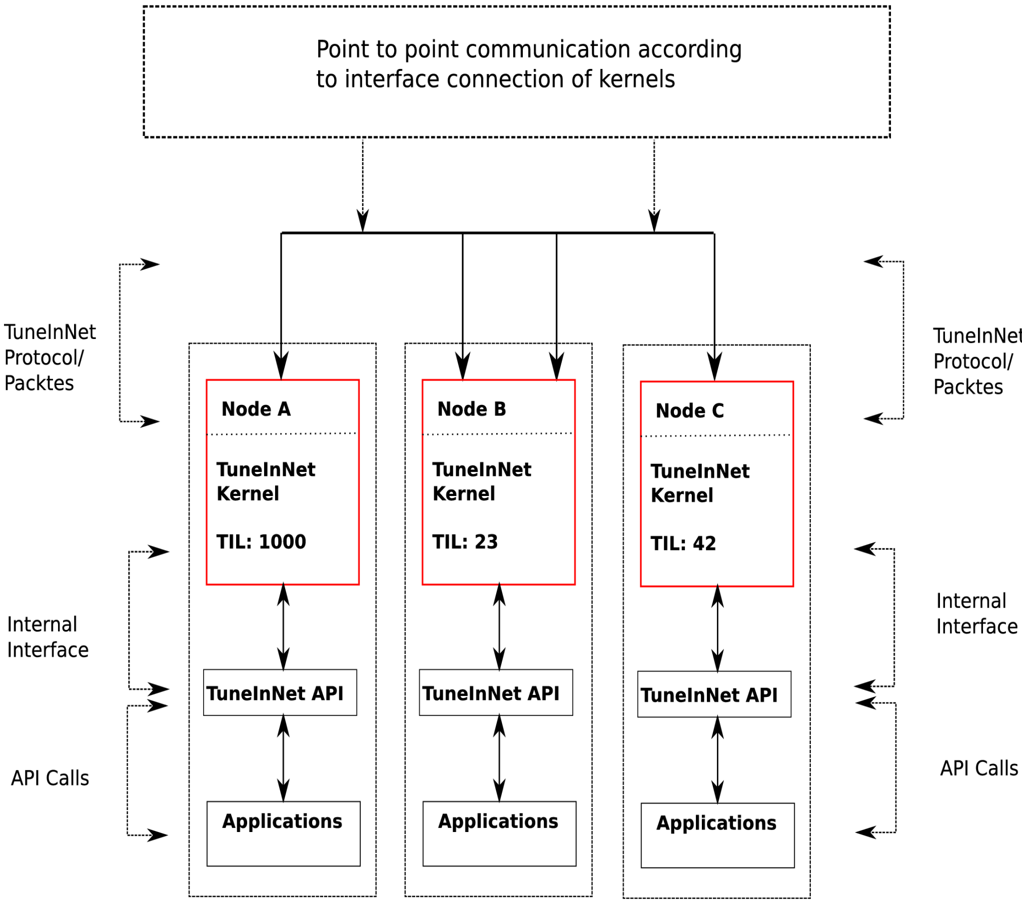


- ϑ is the longest of all shortest paths of a topology, which is generally the reasonable initial value for the TTL

- ϑ is highly important
- ϕ generally increases with |E| and decreases with |V|
- The expected traffic is so high that only favourable topologies seem worth experimenting with
- Rule of thumb is to keep TTL and the eigenvalue of the graph low

Prototype Architecture

Freie Universität Berlin

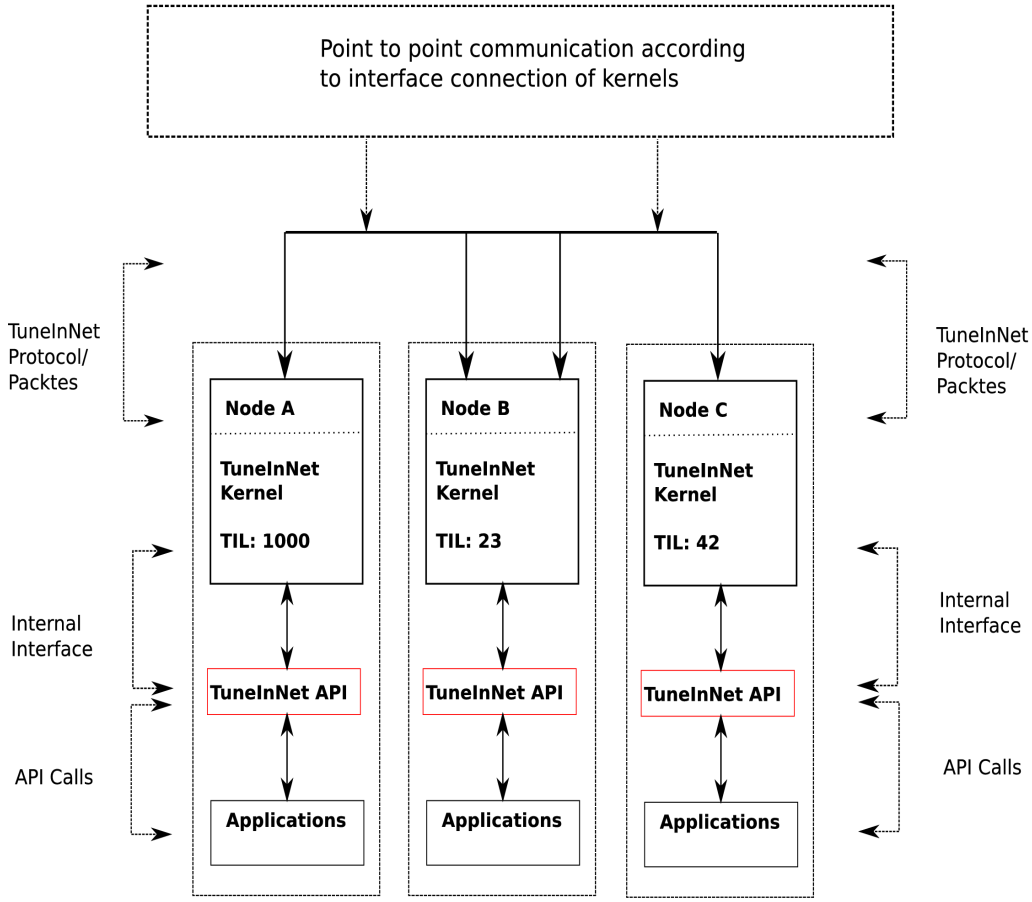


Prototype consists of two parts:

- **TuneInNet Kernel:**
  - **represents an AS**
  - **has unique label TIL**
  - **floods data**
- **TuneInNet API:**
  - **provides methods for applications to communicate over TuneInNet**
  - **uses network wide “ports” (“channels”)**

Prototype Architecture

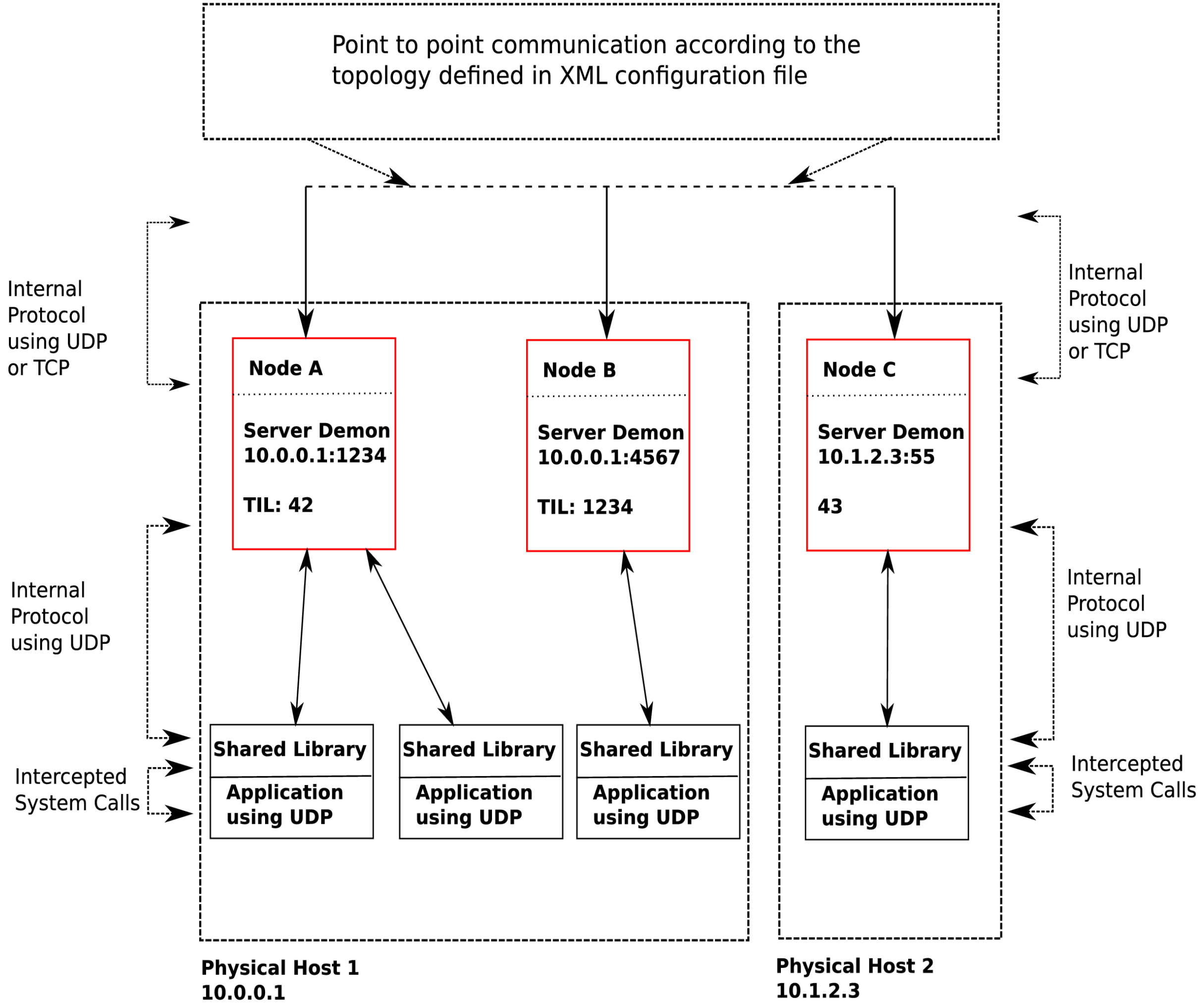
Freie Universität Berlin



Prototype consists of two parts:

- **TuneInNet Kernel:**
  - **represents an AS**
  - **is connected to n peering kernels**
  - **floods data**
- **TuneInNet API:**
  - **provides methods for applications to communicate over TuneInNet**
  - **uses network wide "ports" ("channels")**

Implementation

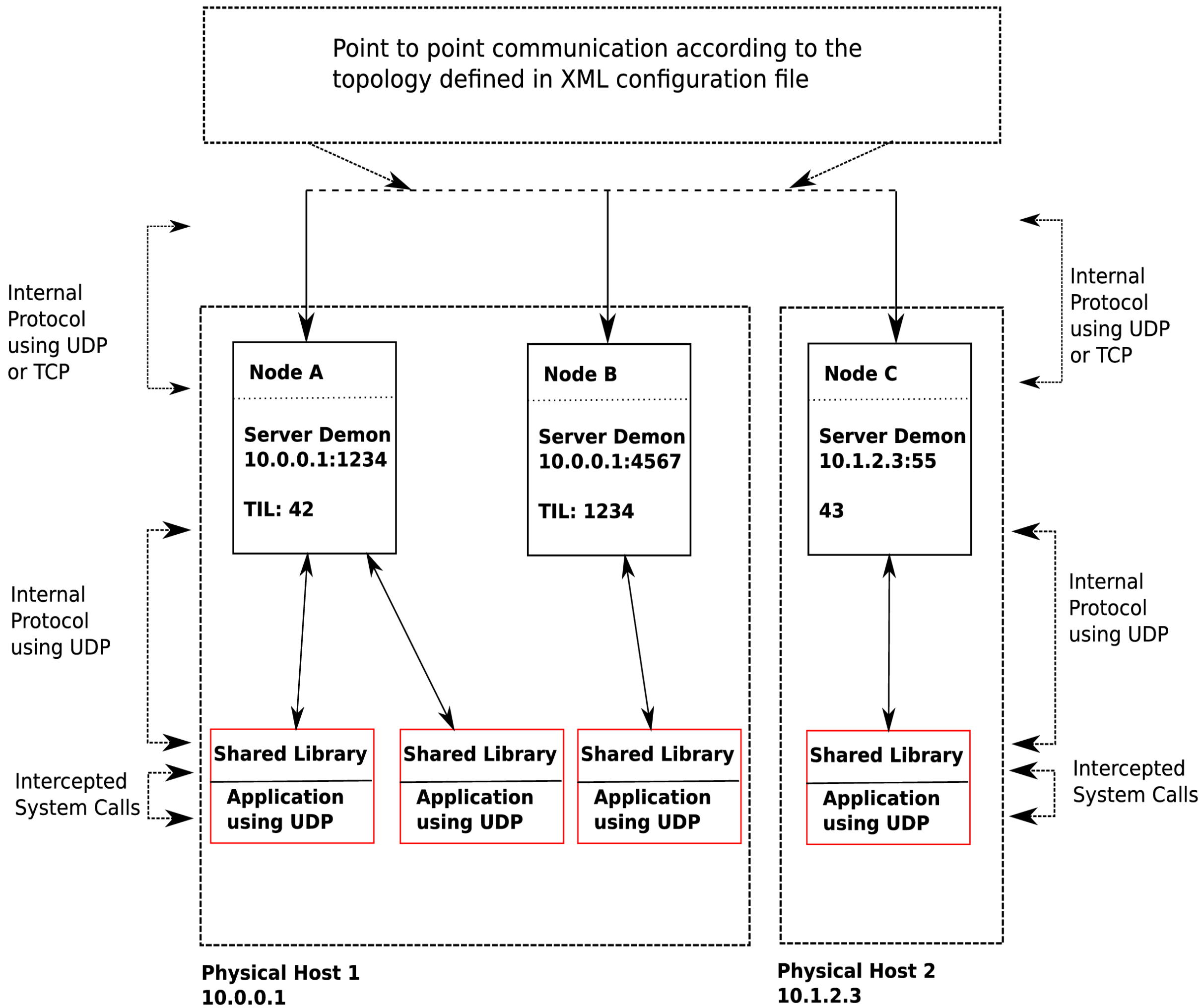


- **Implemented on GNU/Linux with C**
- **Server Daemon implements Kernel**
- **Shared library implements API by offering UDP socket API with new semantics**

➡ allows use of existing applications (iperf,VLC)

Implementation

Freie Universität Berlin



- **Implemented on GNU/Linux with C**
- **Server Daemon implements Kernel**
- **Shared library implements API by offering UDP socket API with new semantics allows use of existing applications (iperf,VLC)**

Evaluation Setup

Freie Universität Berlin

- **Hardware: 20 GNU/Linux hosts (“Russia” and “China”)**

- **Virtual Network Layer**
  - Allows to examine larger networks (~100 nodes)
  - Allows artificial restriction of bandwidth with leaky buckets
  - Includes Reference Multicast Implementation

- **Evaluation Automatization**
  - Topology Generator
  - Scripts

- **Evaluation comprises**
  - Use Case: Video Streaming
  - Jitter
  - Scalability

Use Case – Video Streaming

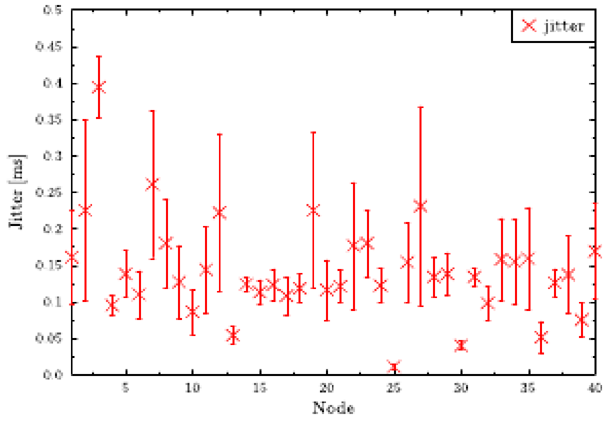
- **VLC Media Player streams video to to what it assumes to be a unicast UDP socket, but really is a TuneInNet port**

- **Several VLC instances “tune in” to this stream**

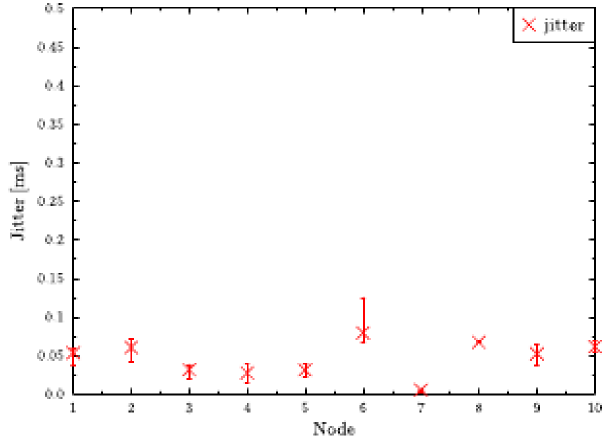
- **The OSD shows that it appears to be the well-known DNS port to the applications**

Jitter

- Jitter (Packet Delay Variation) is important for streaming applications
- Measured with iperf
- Comparison with Reference Multicast Implementation
- Result: Jitter appears considerably higher, possibly because of packet “race conditions”
- Preliminary Finding, because time-critical measurement several layers removed from physical network



TuneInNet



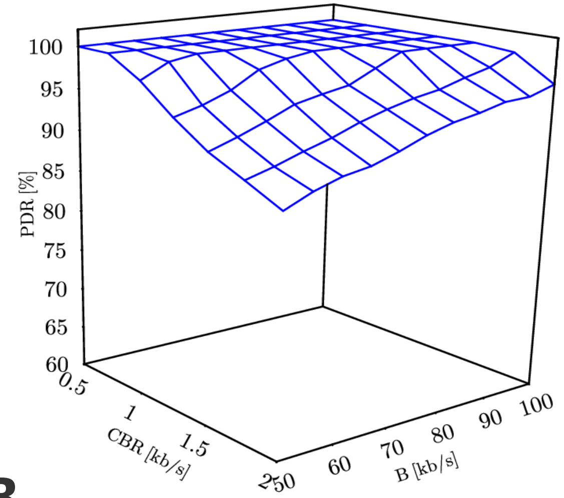
Multicast

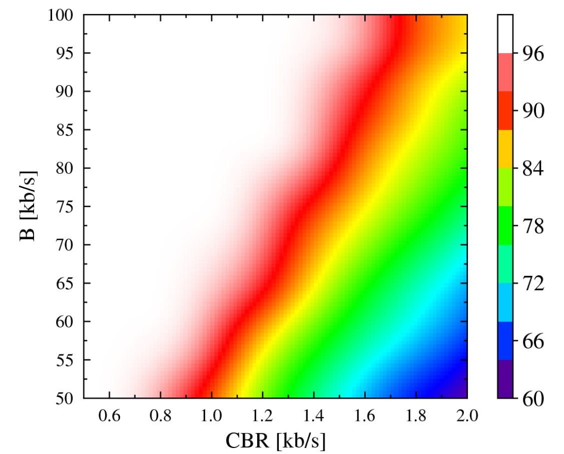
Scalability - Setup

- **Two Topologies, S1 and S2, with sizes of |V|=41 and |V|=100 nodes**
- **Both have favourable attributes, i.e. have low eigenvalue**
- **Set artificial bandwidth to B and send streams with constant bitrate CBR from three different nodes within network.**
- **Measure PDR for all nodes, plot average and worst case PDR**
- **Chose range of values so that PDR lies somewhere around 90%**

Scalability – Topology S1

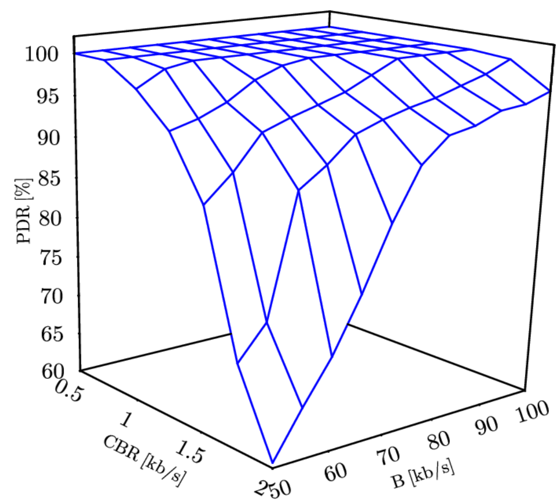
- **Number of nodes |V| = 41**
- **Number of edges |E| = 60**
- **SPL θ(S1) = 6**
- **Total Traffic Modifier Φ(T)=962.4**

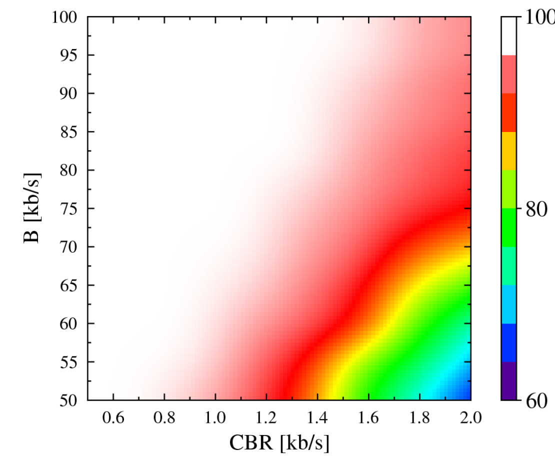




Observations:

- **Drop along increasing CBR and decreasing B**
- **CBR and B seem to have similar influence**
- **Color Map also suggests linear fit**

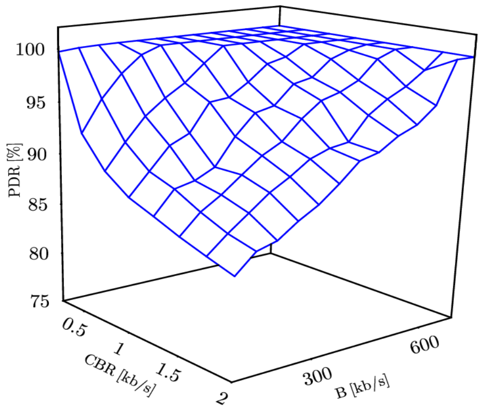


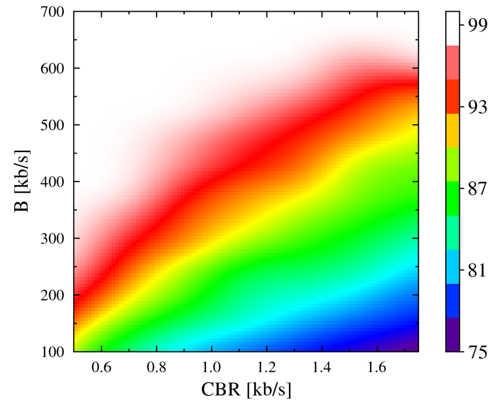


Scalability – Topology S2

Topology S2

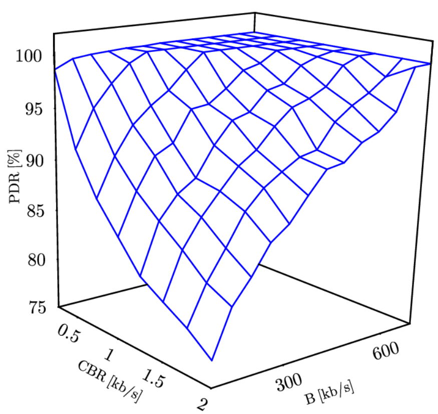
- **Number of nodes |V| = 100**
- **Number of edges |E| = 175**
- **SPL θ(S2) = 8**
- **Total Traffic Modifier Φ(T)=40223**

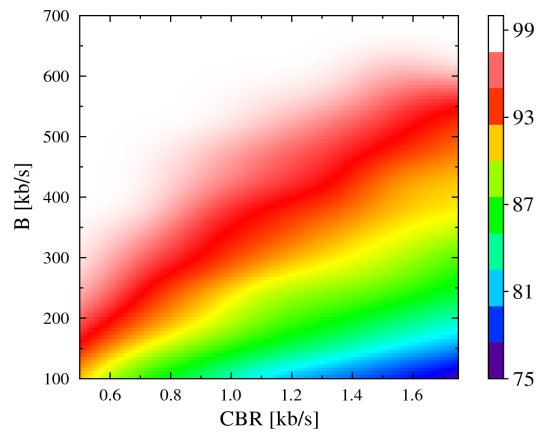




Observations:

- **Less continuous correlations**
- **Color Map still suggests linear fit**





Scalability Summary

Freie Universität Berlin

Linear fits suggest that, for a topology T, there is a constant K, so that for a physical bandwidth B, the effective Bandwidth EB is B/K

EB * K = B

How does K compare to Φ?

$\Phi(T)$	$K(T)$
962.4	40
40223	280

- **K is apparantly not linearly dependent on Φ**
- **K is surprisingly small whith regard to Φ, but is it small enough...?**

Conclusion

- **Radical Version is not economic**

- **Even on a small scale (network of 100 nodes) and with favourable attributes, a 230-fold increase in bandwidth is necessary. On the scale of the Internet, this number would probably increase by orders of magnitude.**

- **Jitter findings, although very preliminary, cast doubt on suitability for multimedia applications despite working use case**

- **Future Work should focus on reducing traffic without introducing complexity:**
  - Filter concept must be rendered more precisely
  - Methods for reducing TTL: dynamic negotiation, deliberate loops in network graphs