Publish Subscribe/CDN

CS525 Class Presentation

Presented By:

Pooja Agarwal and Jayanta Mukherjee

University of Illinois Urbana-Champaign

Publish Subscribe/CDN

- Publish Subscribe Systems
  - Decoupling of publishers and subscribers
  - Greater scalability
  - More dynamic network topology
    - Example: Usenet, OPS
- CDN
  - Replication of data across sites
  - Greater bandwidth of access
    - Example: PPLive, Acamai, Bittorrent

SplitStream: High Bandwidth Multicast in Cooperative Environments

M Castro, P Druschel, A M Kermarrec, A Nandi, A Rowstron, A Singh
SOSP 2003

Presented By: Pooja Agarwal and Jayanta Mukherjee

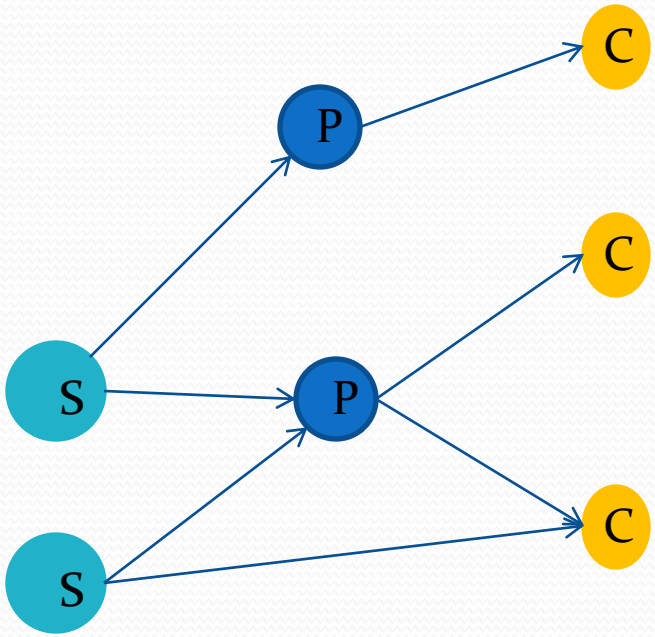
CS 525 Advanced Distributed Systems

Motivation

- Recent Applications
  - IPTV, Tele-conferencing, Tele-immersion
- IP Multicast not widely available
- Why do we need different dissemination system for media rather than reusing file distribution systems?
  - High Bandwidth requirement(typically 1.5Mbps to 100Mbps)
  - Low delay and jitter(<150ms)
  - Periodic streaming(30 to 60 fps)
  - Irregular traffic(I,P,B frames)
  - Instream fault tolerance

Models for Media Streaming

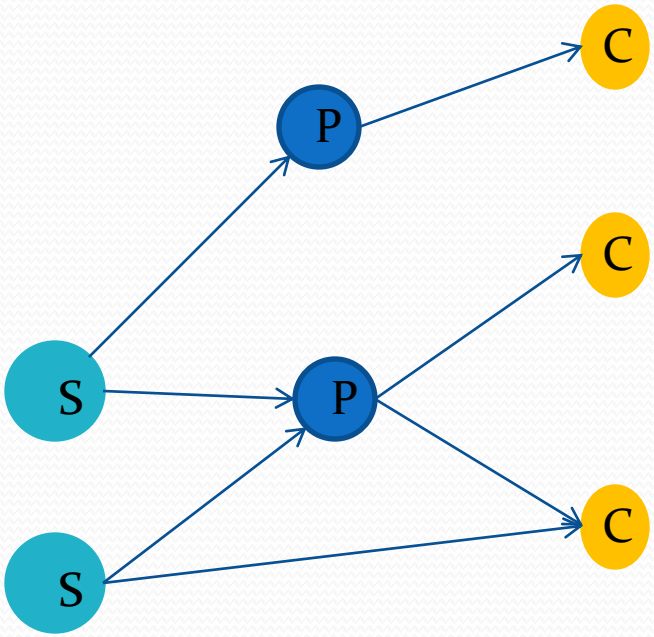
Server-Client Model



- Problems?

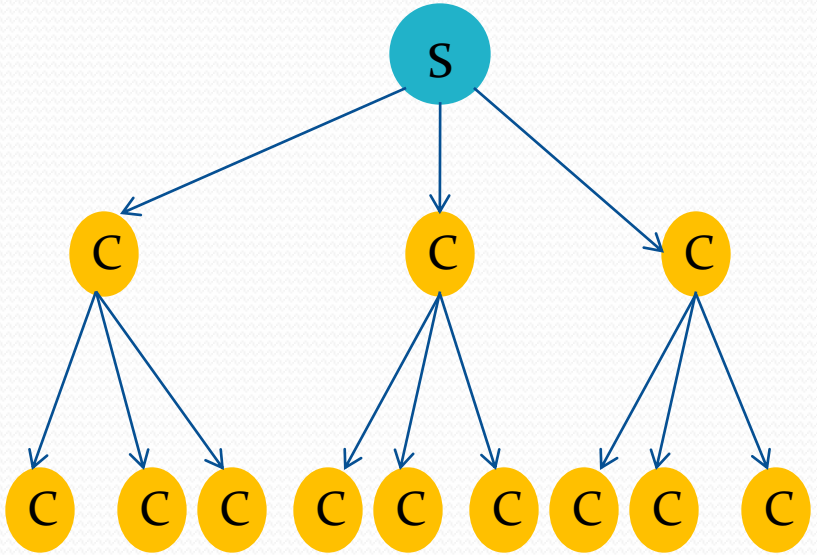
Models for Media Streaming

Server-Client Model



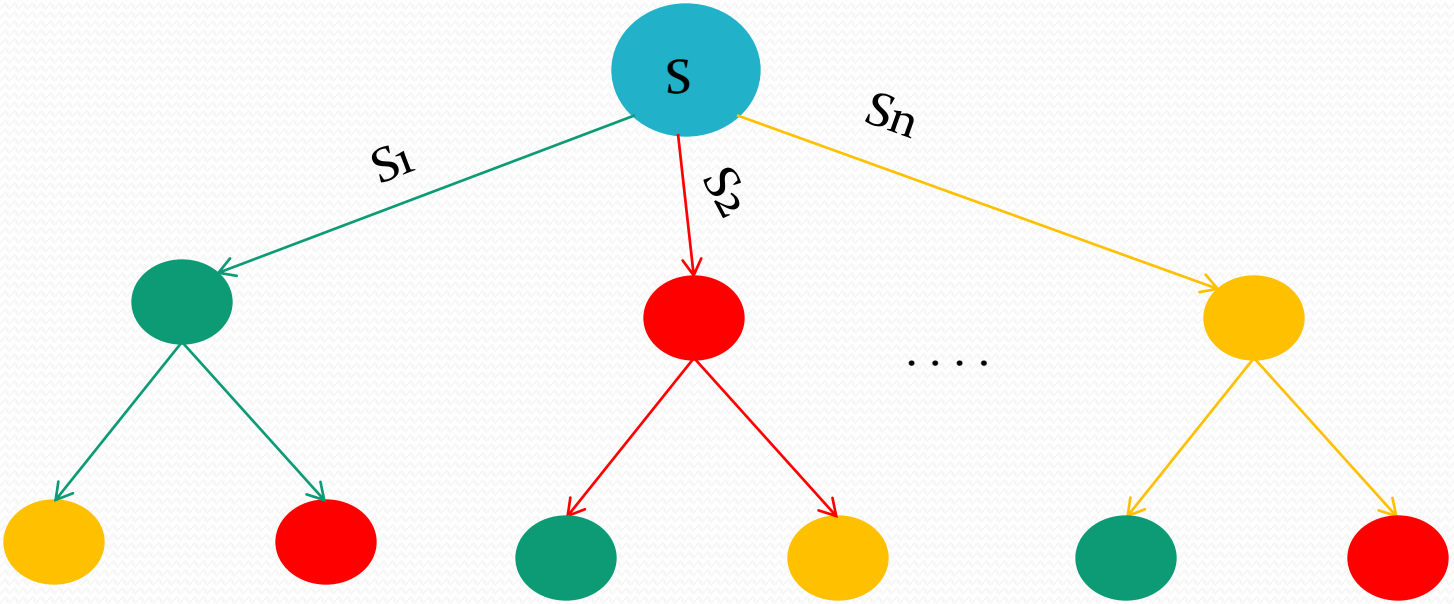
- Problems?

Single Multicast Tree(p2p)



- Number of leaf nodes = f^h, interior nodes = $(f^h - 1)/(f-1)$
- Problems?

SplitStream: Multiple Multicast Trees

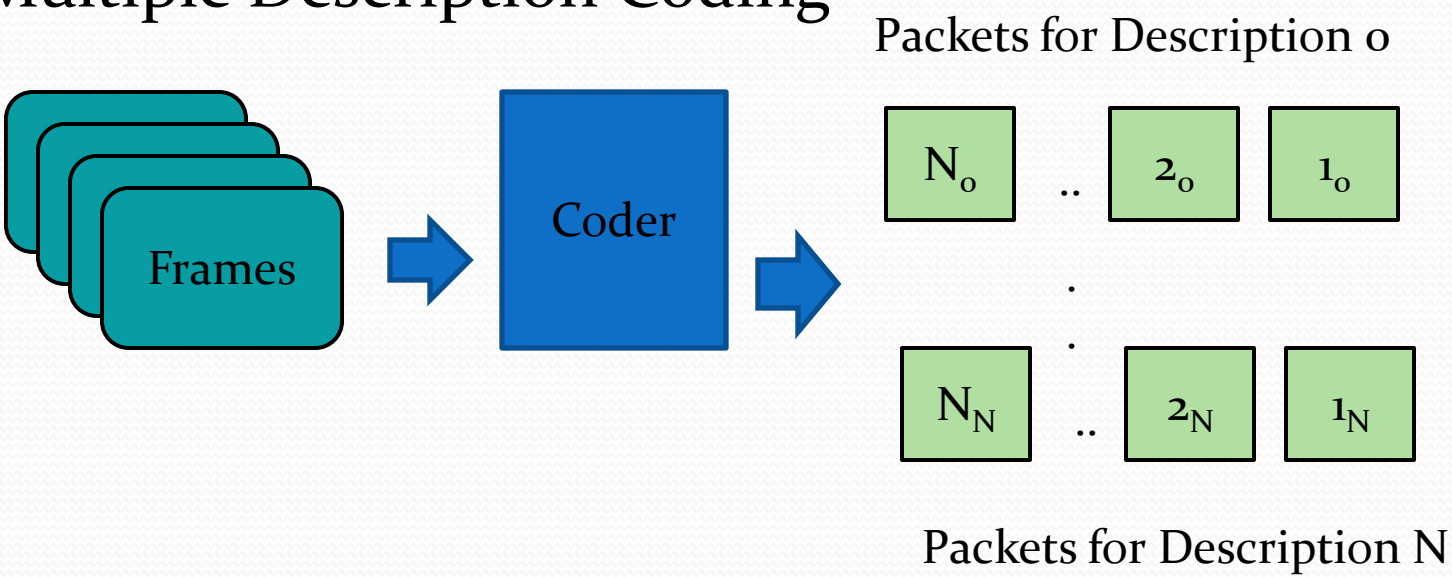


- 2^b = k ensures that forwarding load is balanced
- Inbound bandwidth control through Indegree

❖In this picture, same color represents same node

How to Split Streams?

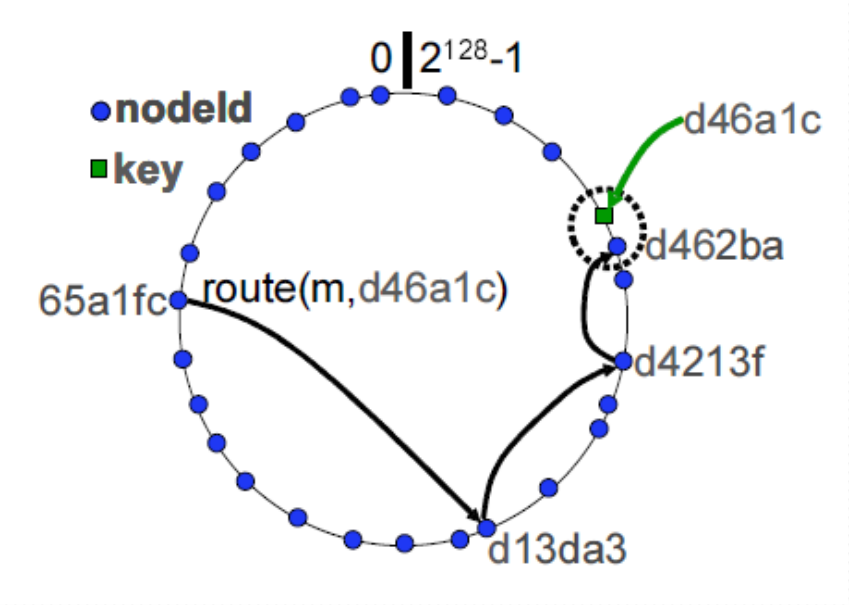
Multiple Description Coding



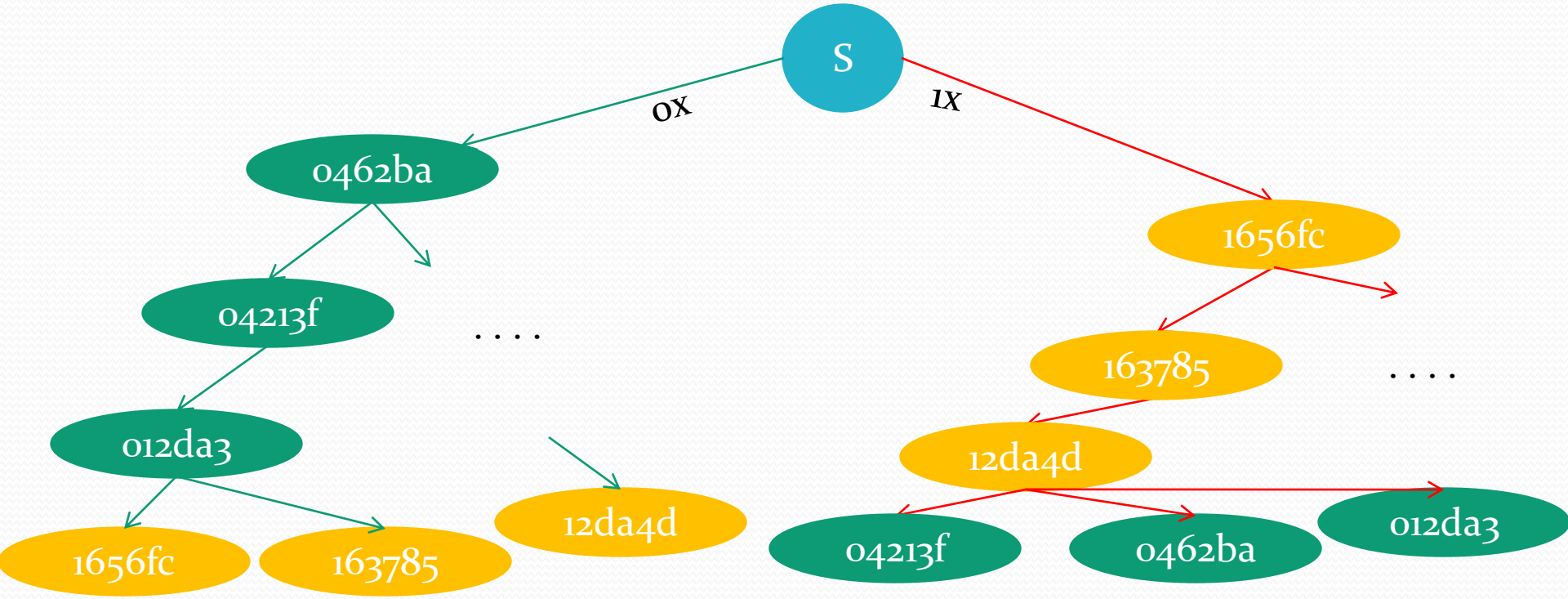
- Each description can be independently decoded
- Is splitting so easy?
  - MPEG-2, MPEG-4

Remember…

- Pastry
  - Routing based on id prefix match



Building Multicast Trees

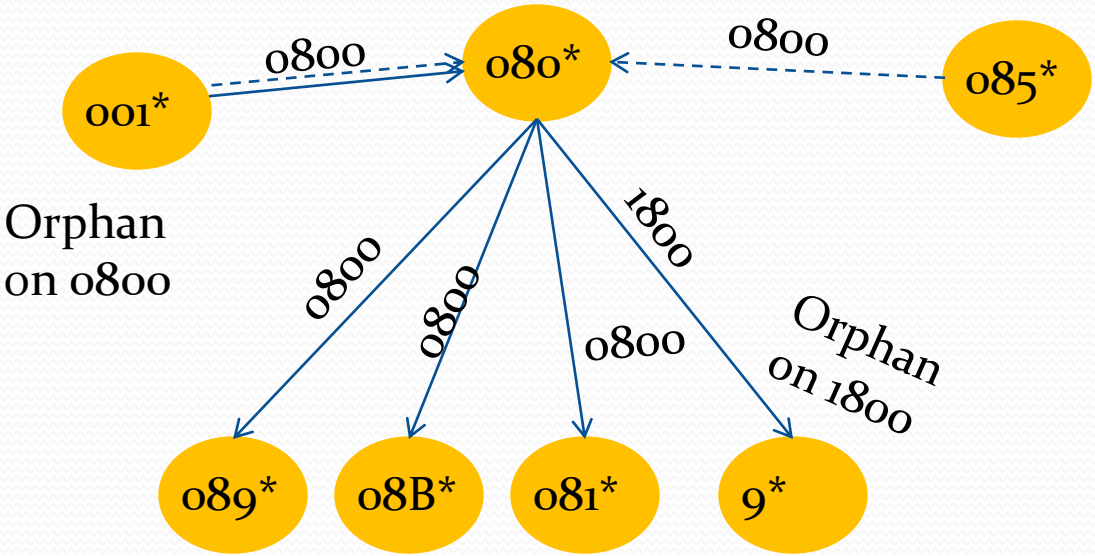


NodeId starting 0x
NodeId starting 1x

- StripeIDs differ in MSB to ensure interior node disjoint trees.
- Reverse path forwarding for tree join

❖In this picture, same color represents same node

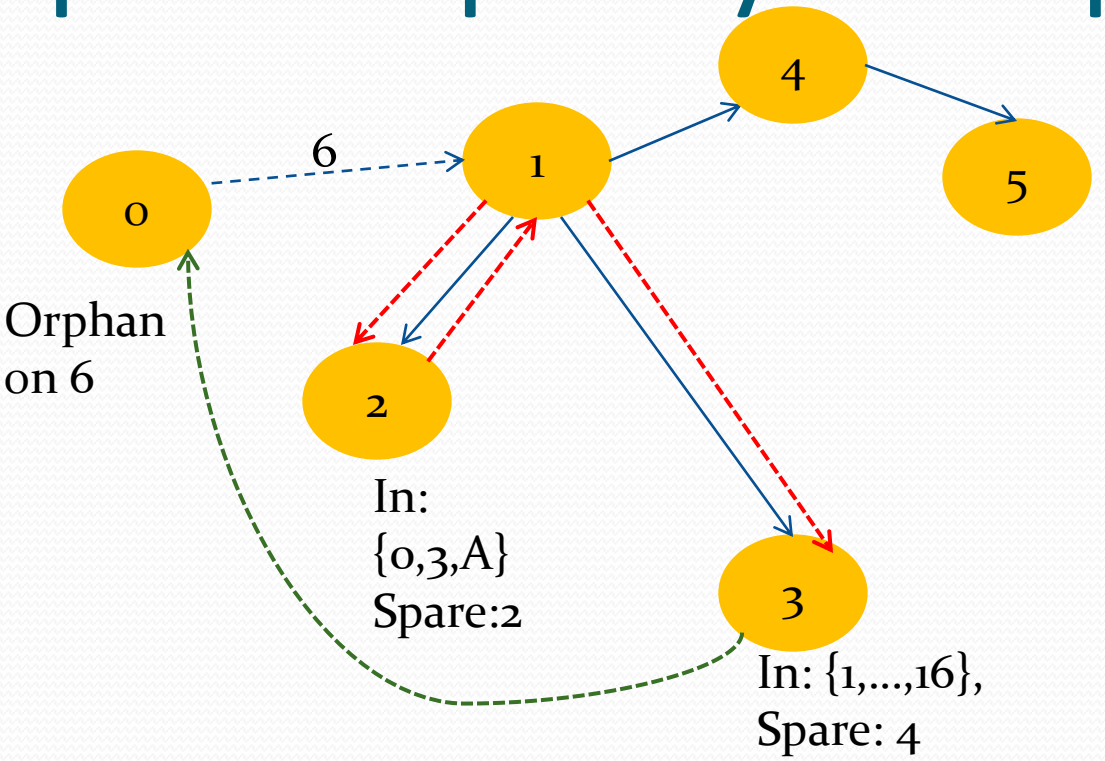
Locating Parents



Orphan performs two steps:

1) Push Down Process
2) Use Spare Capacity group

Spare Capacity Group

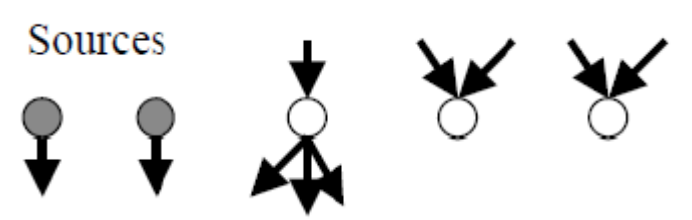


- Anycast
- DFS
- Verify:
  - ✓ stripe available
  - ✓ no cycle formation

- An interior node can become a parent for a streamId which does not share prefix with it's nodeId.
- Fails: no capacity left, desired stripe not available, cycle formation(can be solved)

Is the tree feasible?

- Condition 1:
  - $\Sigma I_i <= \Sigma C_i$
  - Condition 1 is necessary but not sufficient



- Condition 2:
  - Condition 1 holds and
  - For all i: $C_i > I_i$ then $T_i + I_i = k$

Is the tree feasible?(2)

- Probability of failure:

$$|N| \times k \times (1 - \frac{I_{min}}{k})^{\frac{C}{k-1}}$$

  - N = number of nodes
  - K = number of stripes
  - I_{min}= minimum number of stripes that a node receives
  - C = spare capacity = $\Sigma\, C_i$ - $\Sigma\, I_i$
- Success rate is high
- I_{min} is expected to be close to k -> higher success

  - What about free riding?

Complexity

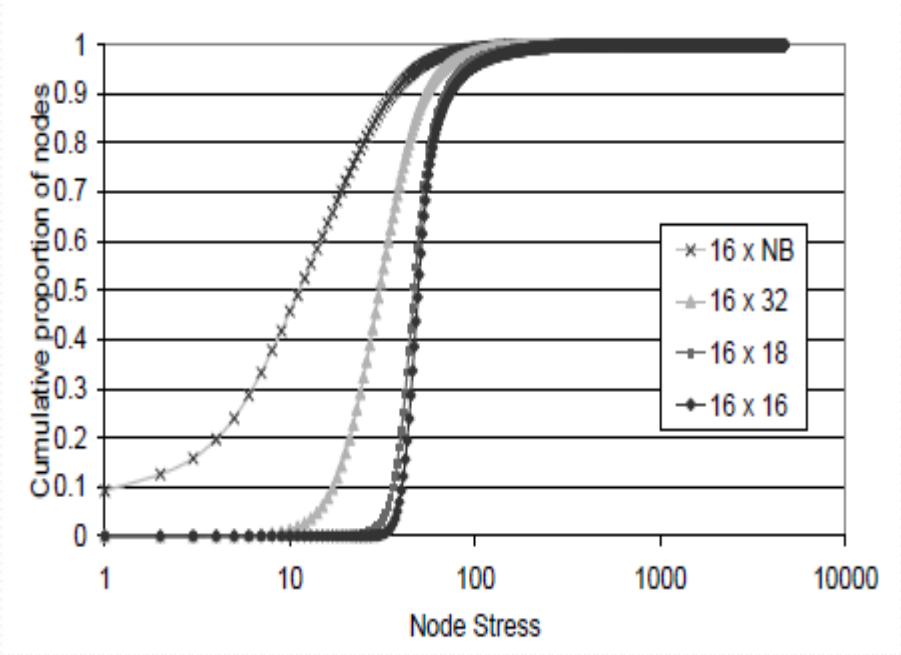
- Expected amount of state maintained by each node = O(*logN*)

- Expected number of messages to build forest = O(*NlogN*) if trees are well balanced, else $O(N^2)$ in worst case.

Experimental Setup

- Simulator models propagation delay.
- Three different network topology model used
  - GATech [5050 routers, 10 transit domain, 10 stub domains, 10 topologies, link delay and routing by graph generator]
  - Mercator[102,639 routers, measurements of internet, 2,662 AS nodes, shortest path routing, no link delay info]
  - CorpNet[298 routers, link delays = minimum of delay over one month period]
- k = 16
- Six Configurations
- Stream size = 320Kbps

Node and Link Stress



Conf.	16 x 16	16 x 18	16 x 32	16 x *NB*	*d* x *d*	Gnut.
Max	1411	1124	886	1616	982	1032
Mean	20.5	19	19	20	11.7	18.2
Med.	16	16	16	16	9	16
Links	.98	.98	.97	.94	.97	.97

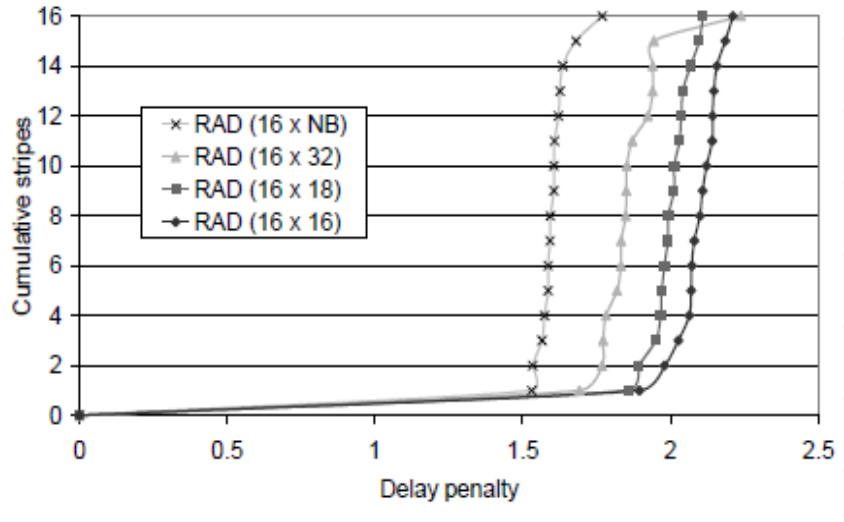
- Link Stress

- 40,000 nodes
- Node stress independent of number of nodes

Forest multicast performance

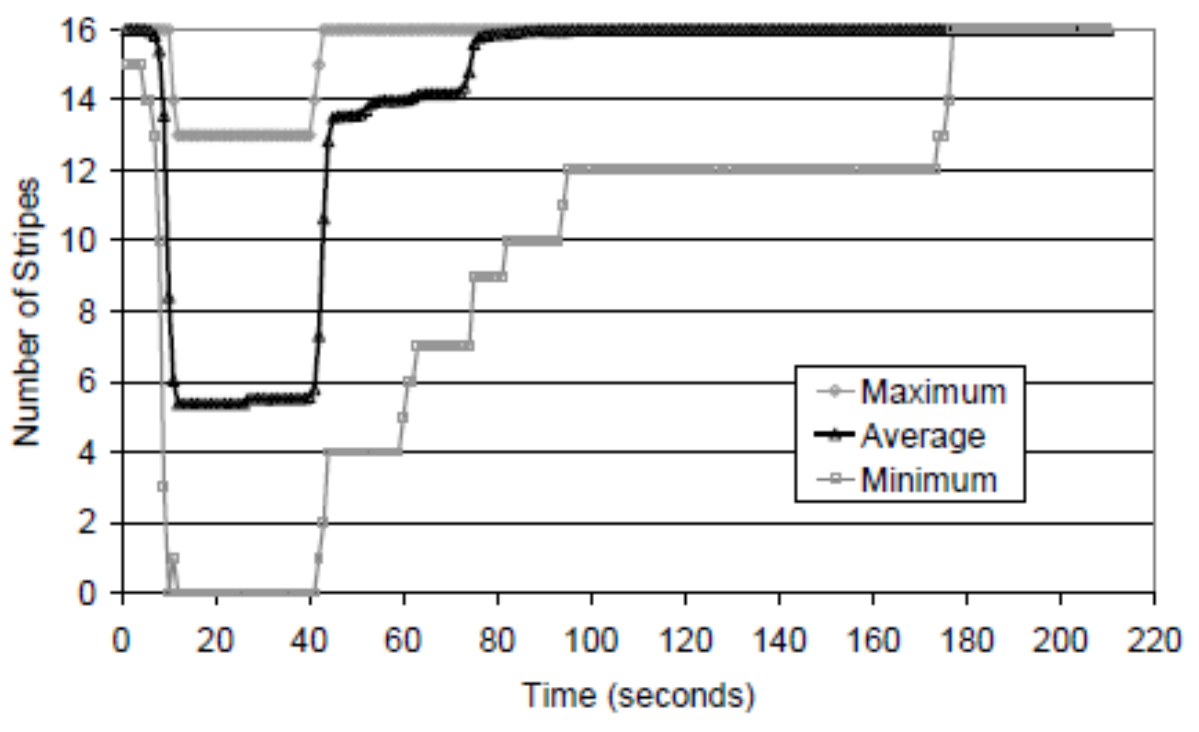
Conf.	centralized	Scribe	IP	16 × 16
Max	639984	3990	16	1411
Mean	128.9	39.6	16	20.5
Median	16	16	16	16
Links	.43	.47	.43	.98

- Link stress: 98% link utilization by splitstream.



- RAD with respect to IP multicast

Catastrophic failures



- 25% out of 10,000 nodes fail

Discussion

- What major problem does SplitStream introduces for multimedia streaming?
  - Synchronization between streams
- How can the synchronization problem be tackled?
  - Bounding delay on receiving all the streams
    - Optimization solution is NP Hard
    - Can Anysee be applied?

Discussion

- Can Splitstream be used as CDN?
  - SplitStream: High Bandwidth Content Distribution in Cooperative Environments, IPTPS'03

CDN

- What is CDN?
  - Content Distribution Network
- CDN replicates the content from origin to the replica servers
- Applications:
  - News Feed
  - Social Networking: Instant Messenger
- Issues with RSS system
  - Causes serious load problems for providers.
  - Workload is “Sticky”
  - Every client periodically checks news source,
    - Consuming significant bandwidth.

Solution

- Content provider impose hard-limits based on IP address
- Trade-off resources for quick update performance

Corona

- Lets look at Corona more closely

CorONA

- Novel, decentralized system for detecting and disseminating Web-page updates
- Solves the load problem
  - Trading off resources for quick update performance
  - Publishers serve content only when
    - Polled involves bandwidth vs update latency
- Operates as a ring of cooperative proxy servers
- Servers dedicated to
  - check the channel and disseminating the news
- Number of servers is determined optimally based on
  - web object popularity, size, and update rate

Corona: A High Performance Publish-Subscribe System for the World Wide Web

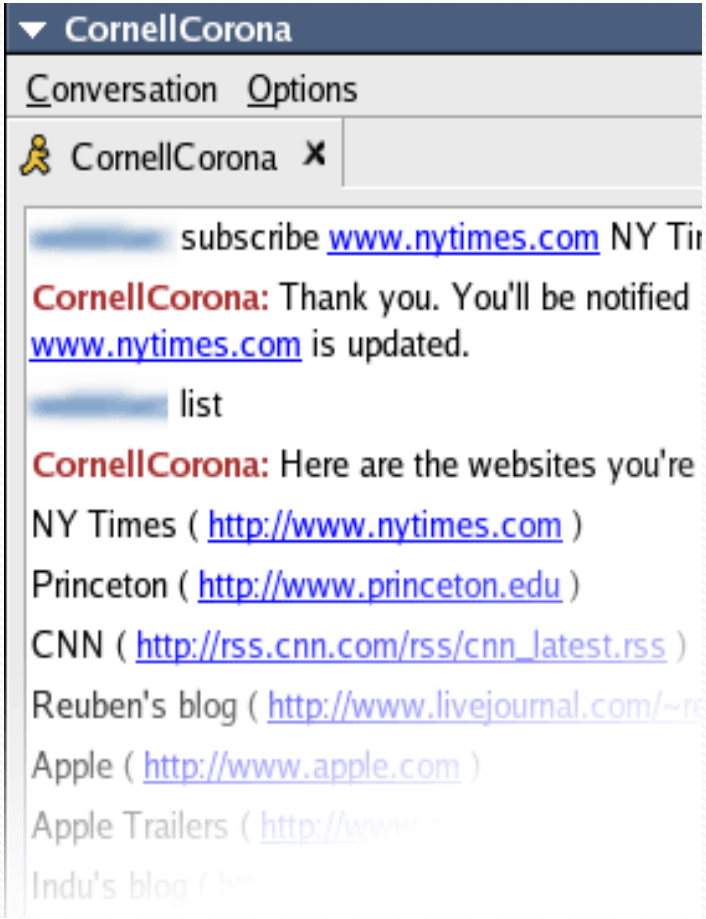
Venugopalan Ramasubramanian Ryan Peterson Emin G¨un Sirer

Cornell University, Ithaca, NY

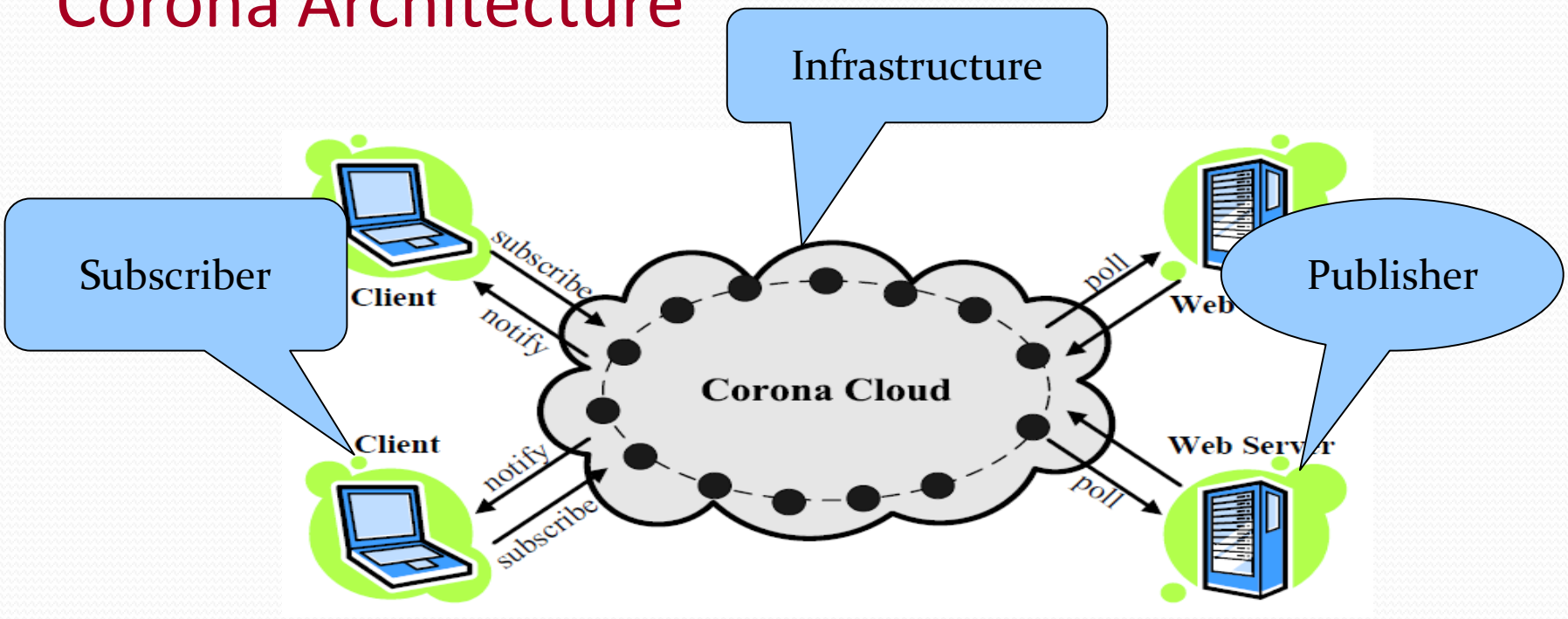
Published at NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation, 2006

CorONA

- Cornell Online News Aggregator
- High-performance publish subscribe system
- Quick and efficient dissemination of web micronews
- It uses Beehive
- interact through instant messages
- backwards compatible with RSS
- RSS: Really Simple Syndication
- Syndication
  - sale of the right to broadcast



Corona Architecture

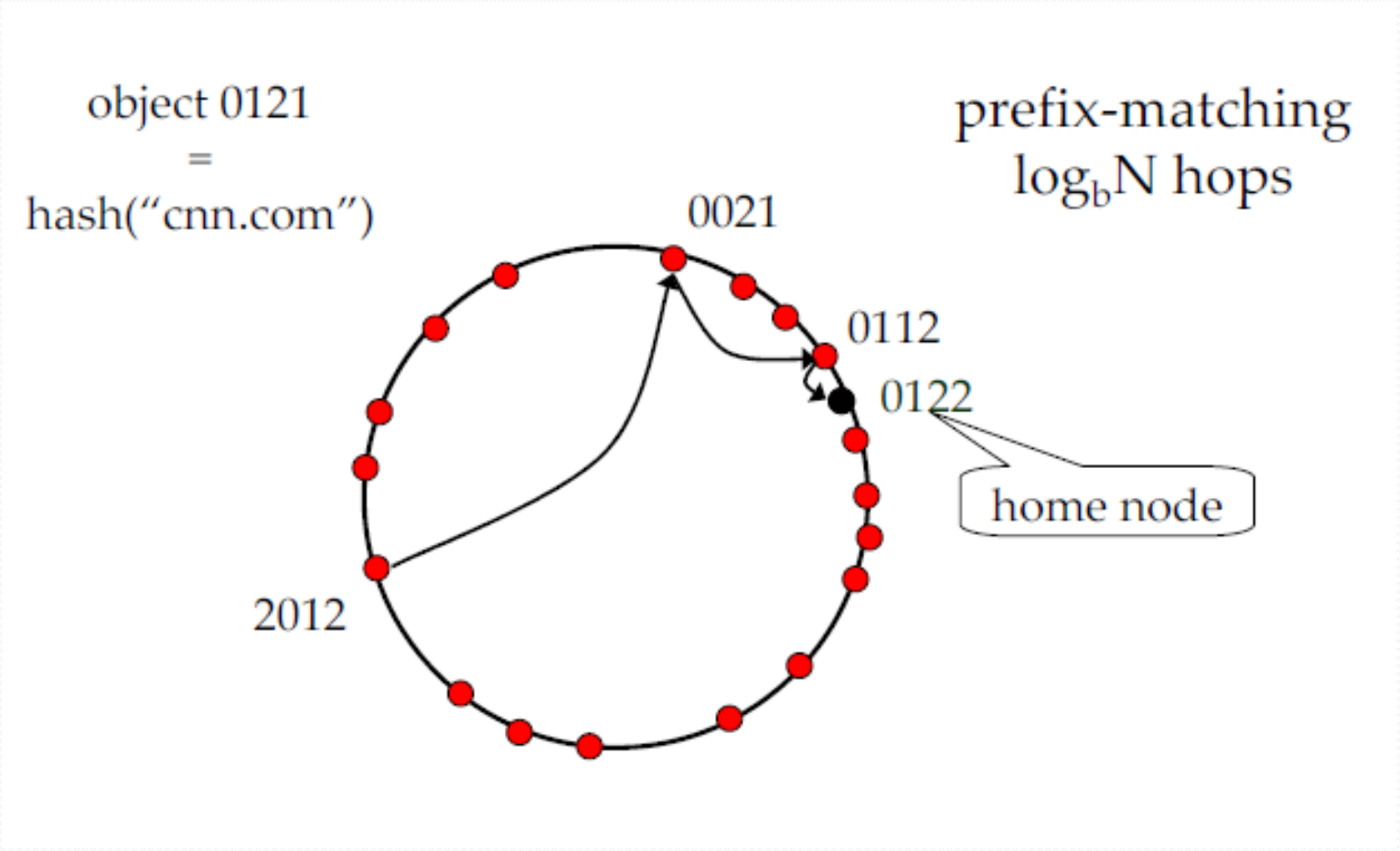


- Decentralized system: nodes act independently share load
- Spreads load uniformly through Consistent-hashing
- Each channel in Corona has a unique identifi er
- *Primary Owner* of a channel is the node with closest identifi er
- Adds additional owners for a channel in order to tolerate failures

System Management

- Corona manages Cooperative Polling through a periodic protocol
- The Periodic protocol consisting of :
- *Optimization phase:*
  - Nodes apply the optimization algorithm on fine-grained tradeoff data for locally polled channels and coarse-grained tradeoff clusters obtained from overlay contacts.
- *Maintenance phase:*
  - Changes to polling levels are communicated to peer nodes in the routing table through maintenance messages
- *Aggregation phase:*
  - Enables nodes to receive new aggregates of tradeoff factors

Structured Overlays: Pastry



Cooperative Polling in Corona

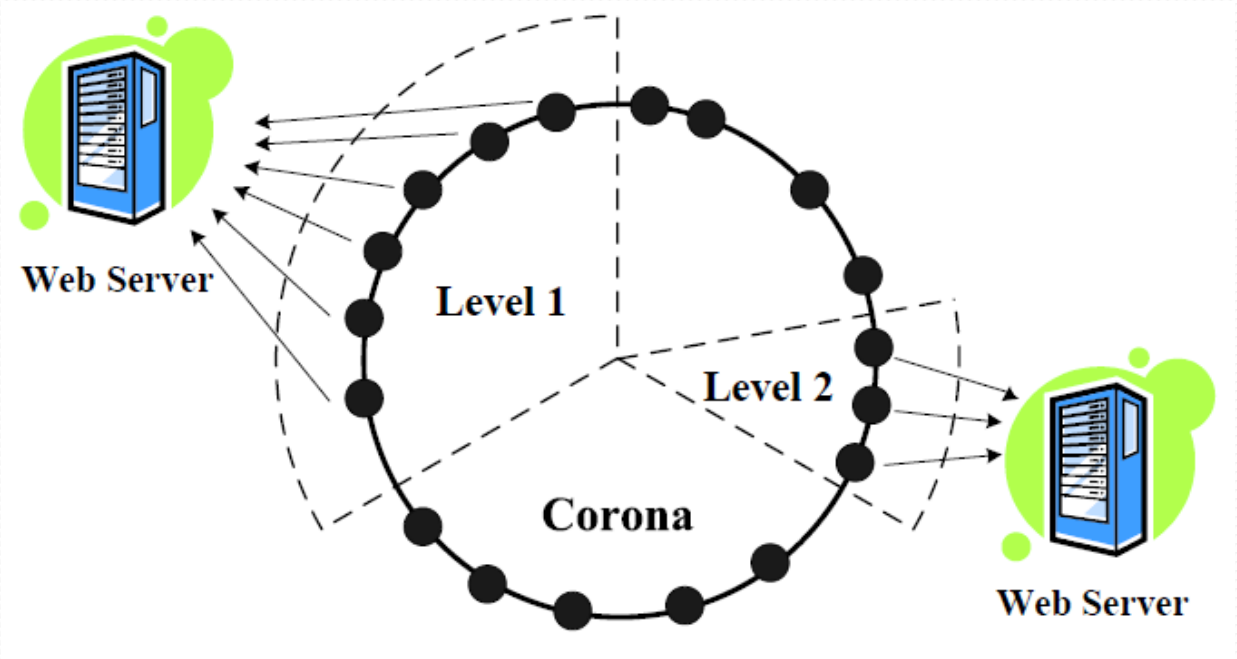


Figure: Each channel is assigned a wedge of nodes to poll the content servers and detect updates. Corona determines the optimal wedge size for each channel through analysis of the global performance overhead Tradeoff. [Figure:2 of Corona-Paper]

Cost-Aware Resource Allocation

- Fundamental cost and performance tradeoff
  - e.g. Lookup latency vs. memory / bandwidth consumption
- System-wide performance goals become constrained optimization problems
- Max. performance s.t. cost ≤ limit
- Min. cost s.t. performance meets target
- Minimize update latency while ensuring the average load on publishers
- Achieve a target update latency while minimize bandwidth consumption

Different Tradeoffs for Optimization

- Corona-Lite
- Corona-Fast
- Corona-Fair
- Corona-Fair-Sqrt
- Corona-Fair-Log

Notations:

τ	polling interval
M	number of channels
N	number of nodes
b	base of structured overlay
T	performance target
l_i	polling level of channel i
q_i	number of clients for channel i
s_i	content size for channel i
u_i	update interval for channel i

- 60 PlanetLab Nodes
- 7500 Channels
- 150K Subscriptions

Corona-Lite

- Minimize the average update detection time while bounding the total network load placed on the content servers.

$$\min. \sum_1^M q_i \frac{b^{l_i}}{N} \qquad \text{s.t.} \sum_1^M s_i \frac{N}{b^{l_i}} \leq \sum_1^M q_i$$

- The overall update performance is measured by taking an average of the update-detection time of each channel weighted by the number of clients subscribed to the channels

Legacy RSS Vs Corona-Lite

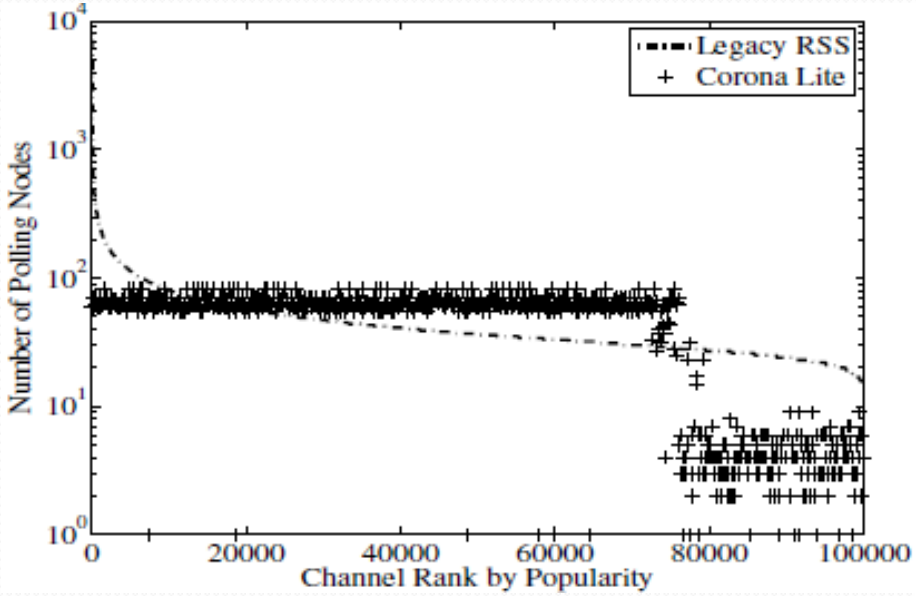


Figure: **Number of Pollers per Channel: Corona trades** off network load from popular channels to decrease update detection time of less popular channels and achieve a lower system-wide average. [Figure-5 of Corona-Paper]

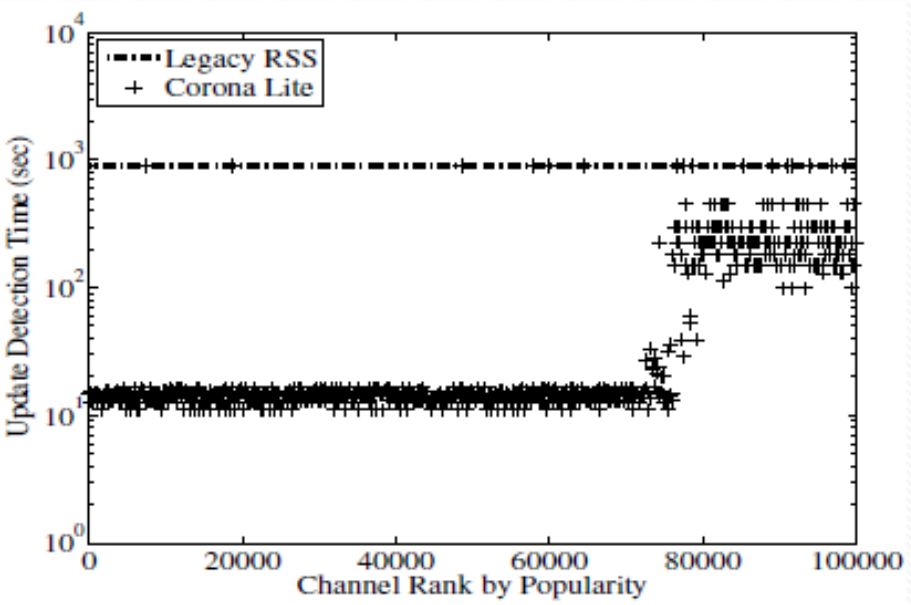


Figure: **Update Detection Time per Channel: Popular** channels gain greater decrease in update detection time **than less popular channels**. [Figure-6 of Corona-Paper]

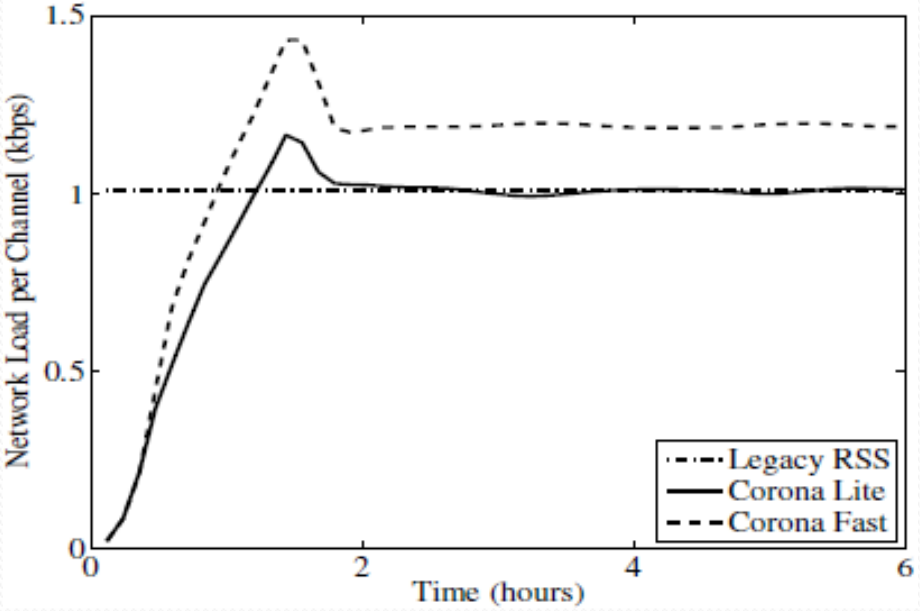
Corona-Fast

- Provides a stable update performance
- Steady performance at a desired level through changes in working load
- Minimizes total network-load on the content servers while meeting a target average update detection time.

$$\text{min.} \sum_1^M s_i \frac{N}{b^{l_i}} \qquad \text{s.t.} \sum_1^M q_i \frac{b^{l_i}}{N} \leq T \sum_1^M q_i$$

- It enables us to tune the update performance of the system according to application needs.

Legacy RSS Vs Corona-{Lite/Fast}



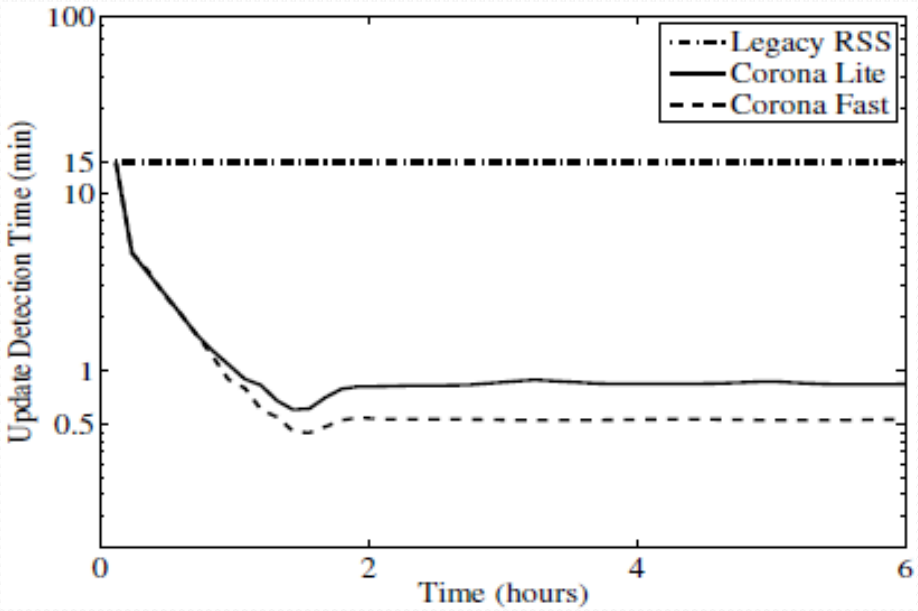


Figure: Network Load on Content Servers: Corona-Lite converges quickly to match the network load imposed by legacy RSS clients [Figure-3 of Corona-Paper]

Figure: Average Update Detection Time: Corona-Lite provides 15-fold improvement in update detection time compared to legacy RSS clients for the same network load. [Figure-4 of Corona-Paper]

Limitations of Corona-{Lite/Fast}

- Do not consider the actual rate of change of content in a channel.
- While some Web-objects are updated every few minutes, others do not change for days at a time

Solution????

Corona-Fair

Corona-Fair

- Corona-Fair incorporates the update rate of channels into the performance tradeoff in order to achieve a fairer distribution of update performance between channels.
- Minimize average update detection time w.r.t. expected update frequency, bounding load on content servers

$$\text{min.} \sum_1^M q_i \frac{\tau}{u_i} \frac{b^l i}{N} \qquad \text{s.t.} \sum_1^M s_i \frac{N}{b^l i} \leq \sum_1^M q_i$$

- De nies a modi ed update performance metric as the ratio of the update detection time and the update interval of the channel, which it minimizes to achieve a target load.
- Biases the performance unfavorably against channels with large update interval times.

Corona-Lite Vs Corona-Fair

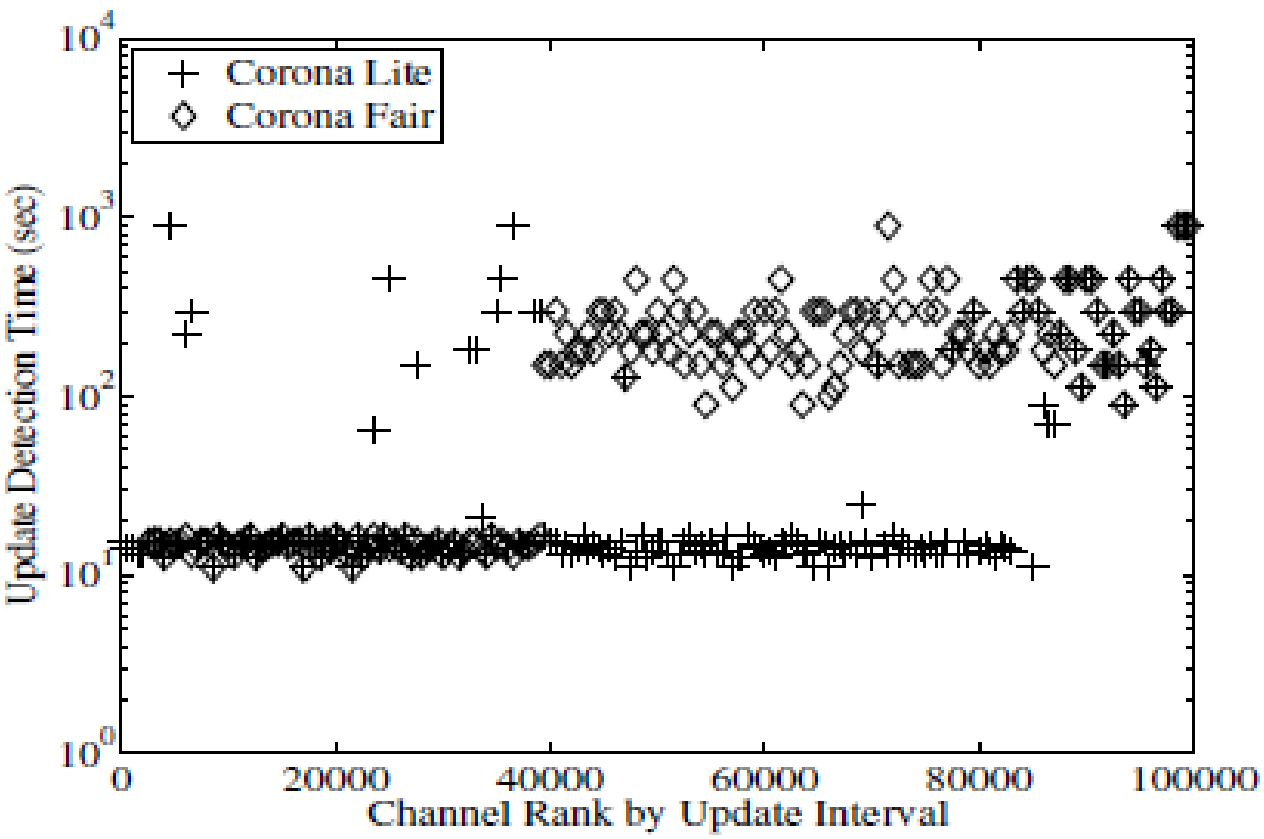


Figure: **Update Detection Time per Channel: Corona-**Fair provides better update detection time for channels that change rapidly than for channels that change rarely. [Figure-7 of Corona-Paper]

Issues with Corona-Fair

A channel that does not change for several days experiences long update detection times, even if there are many subscribers for the channel.

Solution????

- **Compensate for this bias**
  - **Update performance metrics based on sq.root and log**

WHY???

- Square root and Logarithmic functions grow sub-linearly
- Sub-linear metric dampens the tendency of the optimization algorithm to punish slow-changing yet popular feeds.

Corona-Fair-Sqrt & Corona-Fair-Log

Corona-Fair-Sqrt & Corona-Fair-Log

- **Corona-Fair with sqrt weight on the latency ratio to emphasize infrequently changing channels.**

$$\text{min.} \sum_1^M q_i \frac{\sqrt{\tau}}{\sqrt{u_i}} \frac{b^l{}_i}{N} \quad \text{s.t.} \sum_1^M s_i \frac{N}{b^l{}_i} \leq \sum_1^M q_i$$

- **Corona-Fair with log weight on the latency ratio to emphasize infrequently changing channels.**

$$\text{min.} \sum_1^M q_i \frac{\log \tau}{\log u_i} \frac{b^l{}_i}{N} \quad \text{s.t.} \sum_1^M s_i \frac{N}{b^l{}_i} \leq \sum_1^M q_i$$

Corona-Fair-Sqrt Vs Corona-Fair-Log

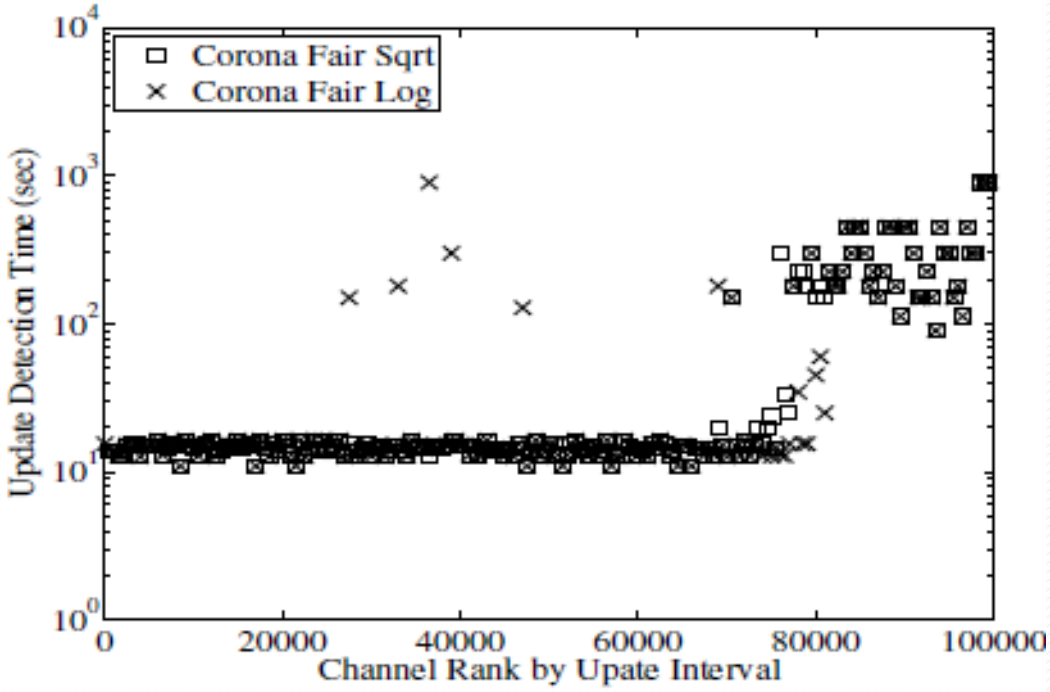


Figure: **Update Detection Time per Channel: Corona-**Fair-Sqrt and Corona-Fair-Log ?x the bias against channels that change rarely and provide better update detection [Figure 8 of Corona Paper]

Legacy RSS Vs Corona

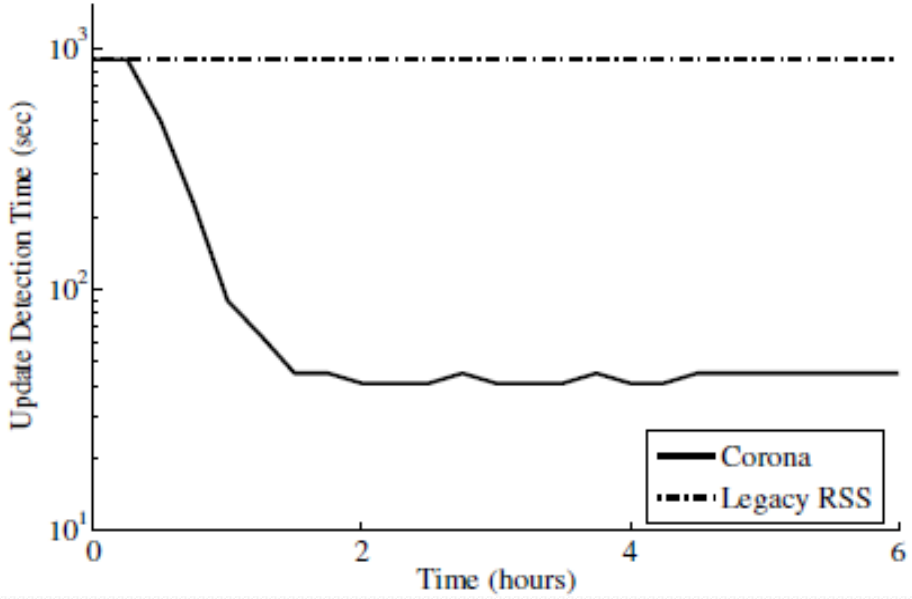


Figure: **Average Update Detection Time: Corona provides** an order of magnitude lower update detection time **compared to legacy RSS.**[Figure-9 of Corona-Paper]

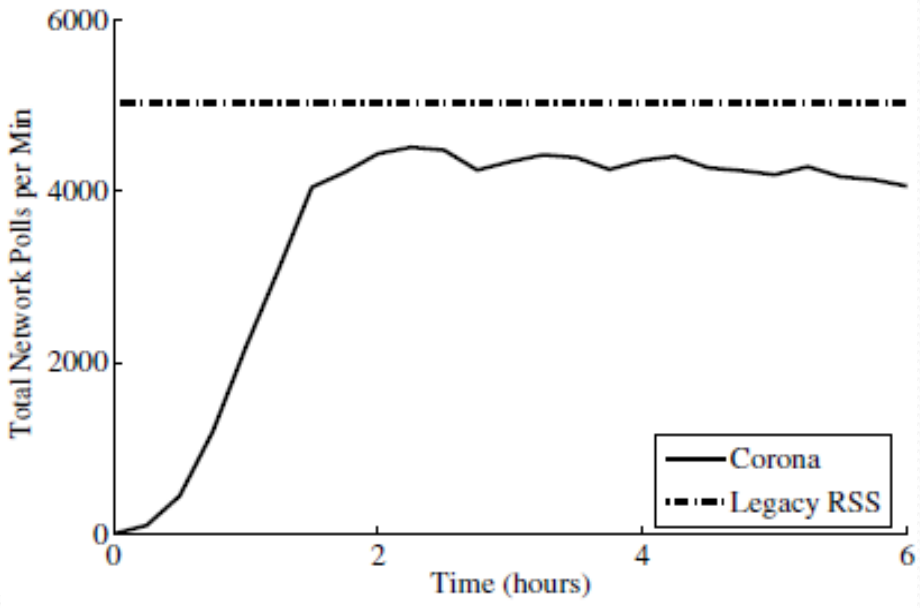


Figure: **Total Polling Load on Servers: The total load** generated by Corona is well below the load generated by **clients using legacy RSS** [Figure-10 of Corona-Paper]

Performance Summary

Scheme	Average Update Detection Time (sec)	Average Load (polls per 30 min per channel)
Legacy-RSS	900	50.00
Corona-Lite	53	48.97
Corona-Fair	142	50.14
Corona-Fair-Sqrt	55	49.46
Corona-Fair-Log	53	49.43
Corona-Fast	32	58.75

More realistic

Fastest

Discussions

- Does not require any change in the content sources
- Globally optimum allocation of bandwidth
- Extensive Simulation and practical results
- Shield web-servers from sudden increase in load
- Suitable for Pull-based architecture
- The average update time is 45 Sec
- Is this model suitable for Stock-Market?

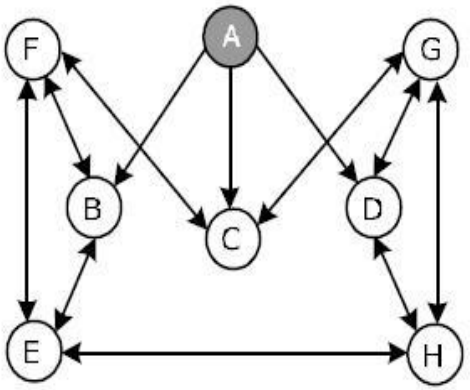
AnySee: Peer-to-Peer Live Streaming

Xiaofei Liao, Hai Jin, Yunhao Liu, Lionel M. Ni, and Dafu Deng

IEEE INFOCOM 2006, Barcelona, Spain, April 2006

Mesh-based Overlay

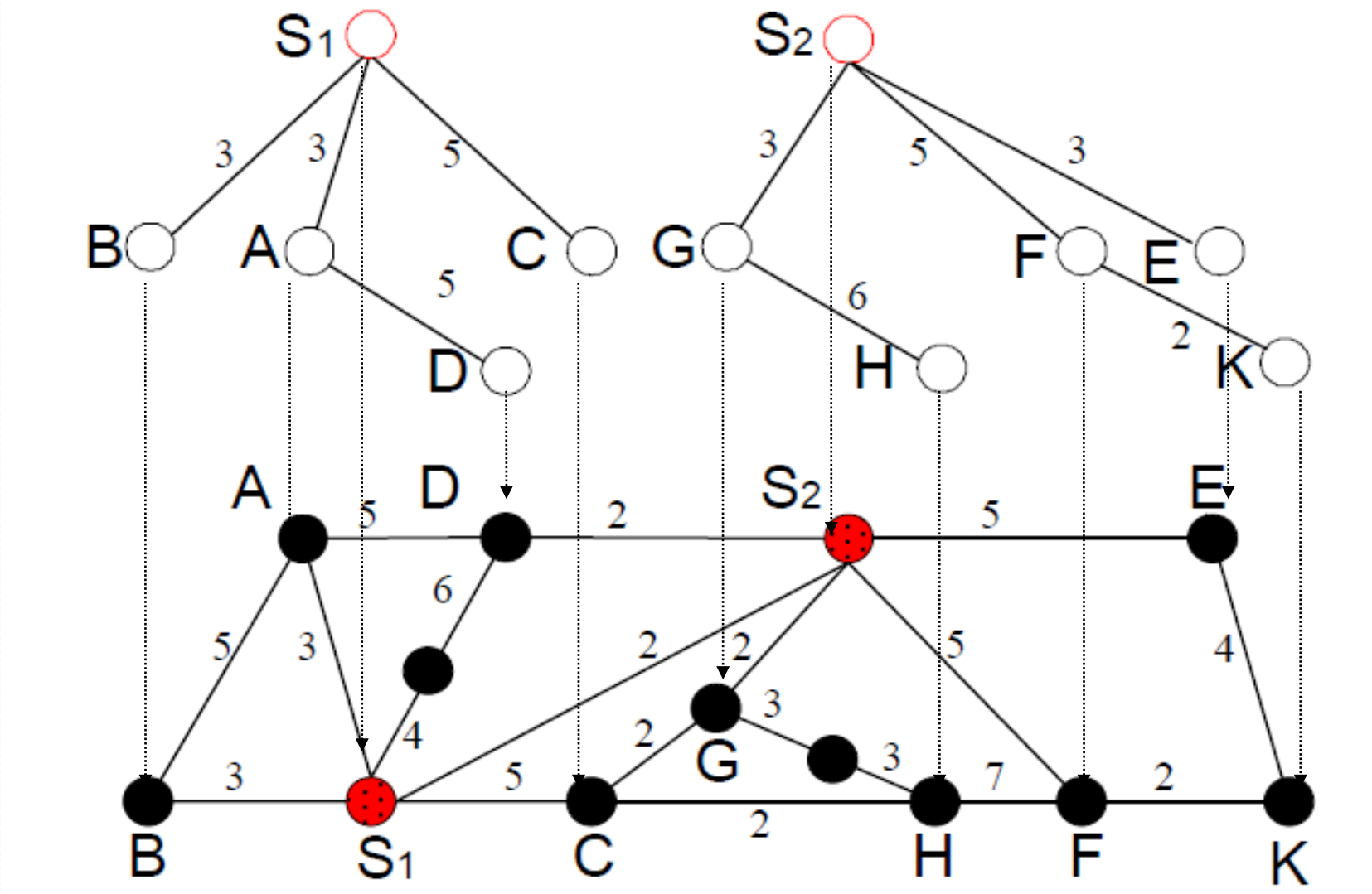
- **Each peer can accept media data from multiple parents as well as provide services to multiple children**
- **Example: Coolstreaming , Promise, GNUStream**
-
- **Pros**:
  - High resource utilization
  - Fast discovery of fresh peers due to gossiping
- **Cons**:
  - Quality of service cannot be guaranteed due to gossiping
  - large buffer space needed to reduce impact of autonomy of peers (in a dynamic environment)



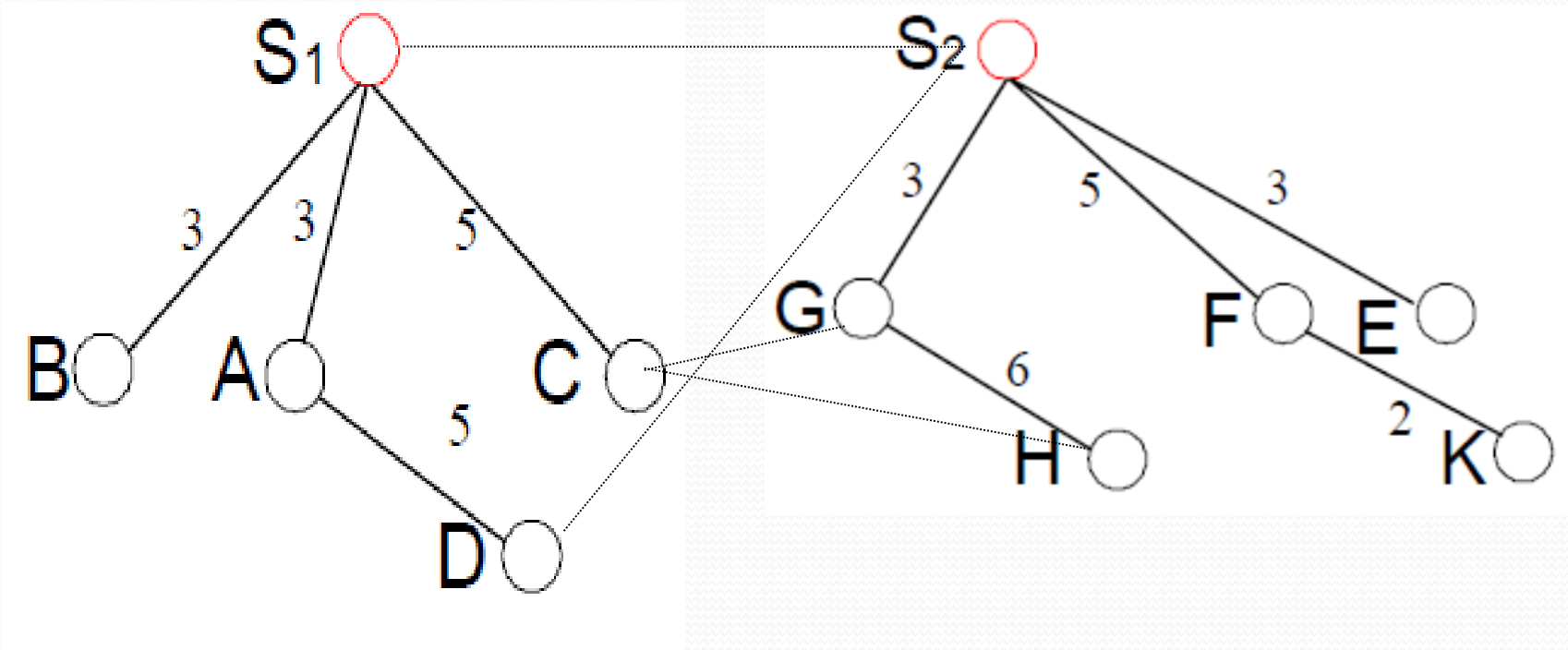
AnySee

- **A peer-to-peer efficient, scalable live streaming system**
  - adopts an inter-overlay optimization scheme
- **Objective:**
  - To improve global resource utilization and distribute traffic evenly
  - Assign resources based on their locality and delay
  - Assure streaming service quality by using the nearest peers from different overlays
  - Balance the load among the group members.
- Released in 2004 in CERNET
- 60000 users: TV, Movies, academic conferences

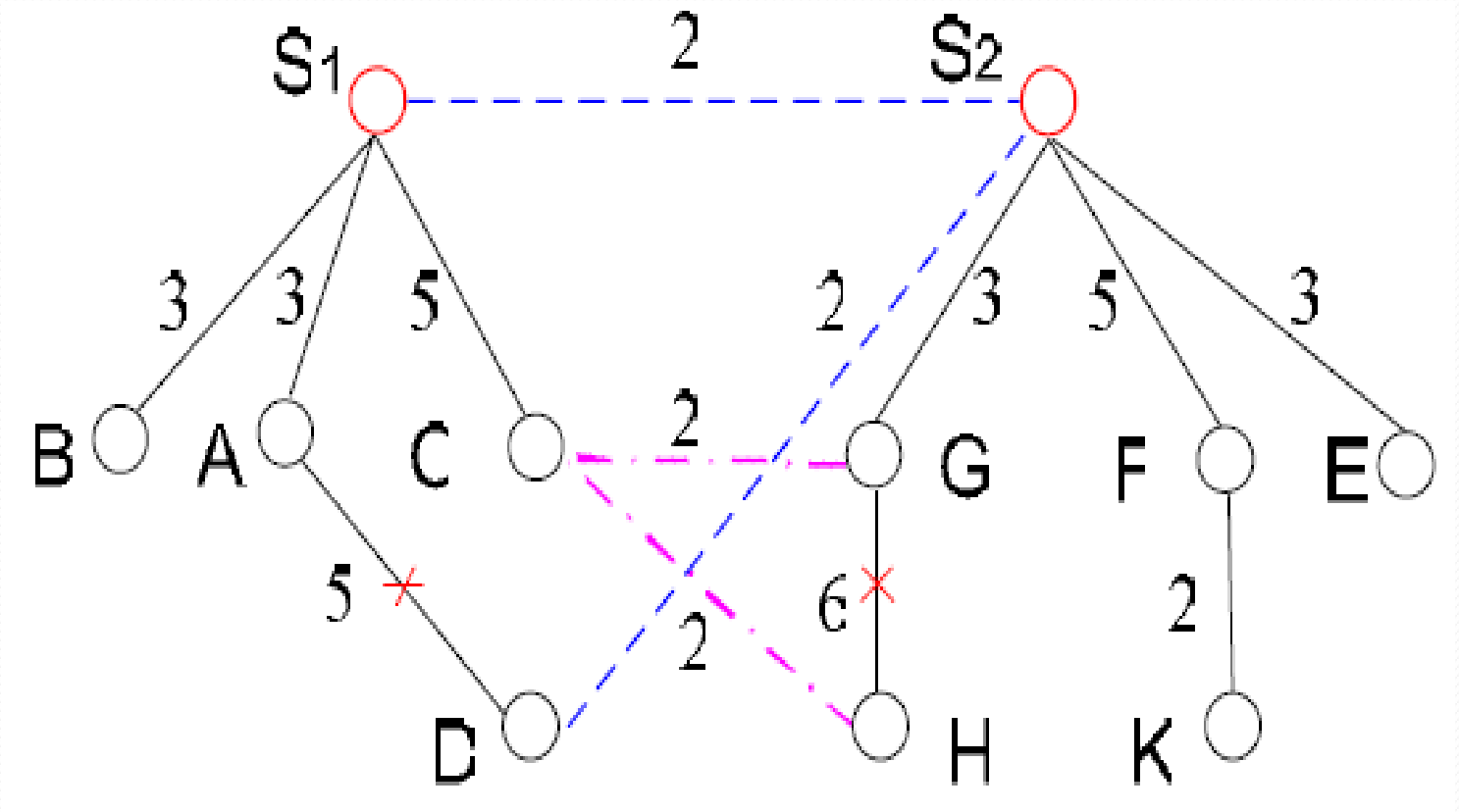
Intra-Overlay Optimization



AnySee Inter-Overlay Optimization



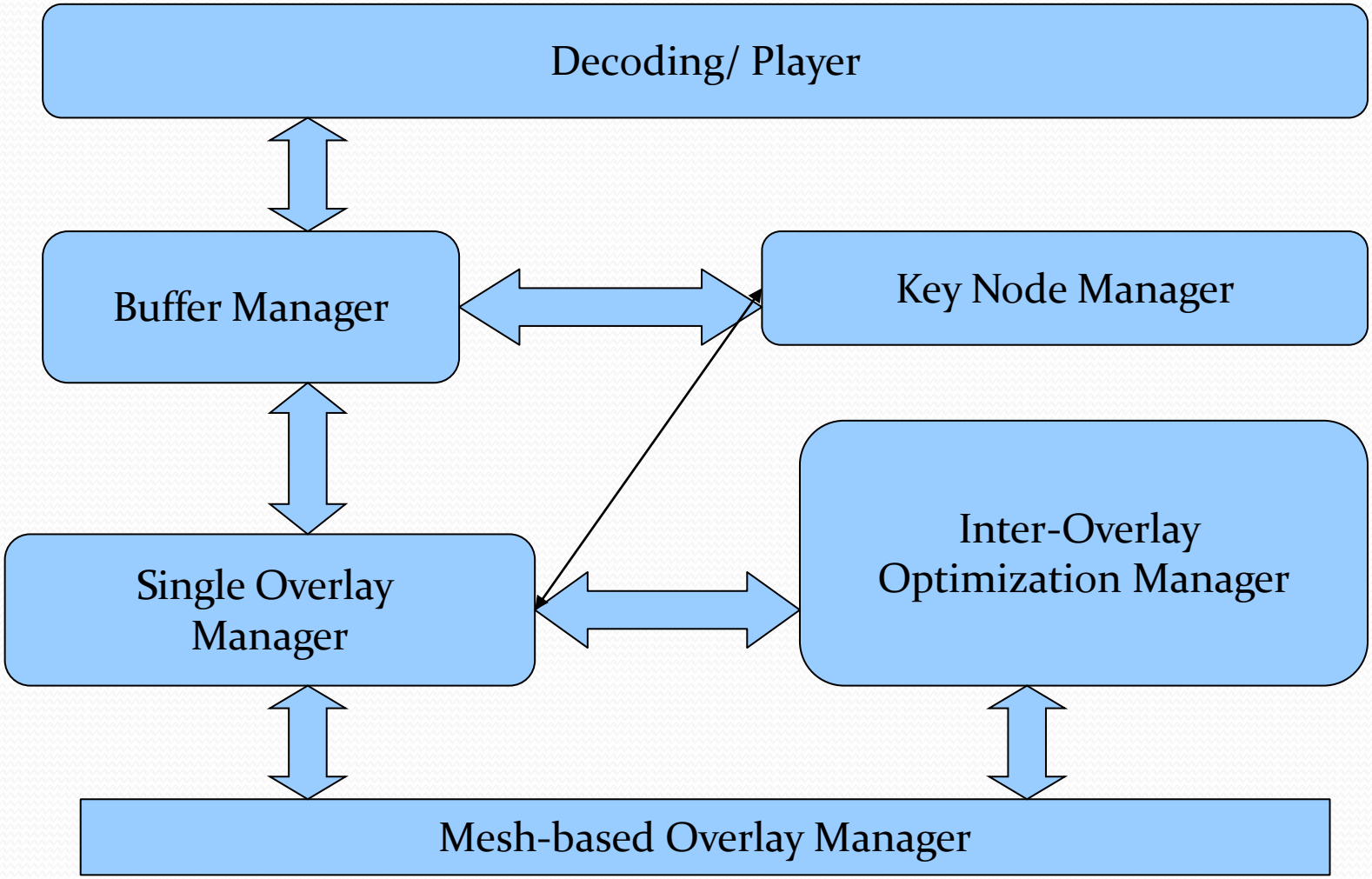
AnySee Inter-Overlay Optimization



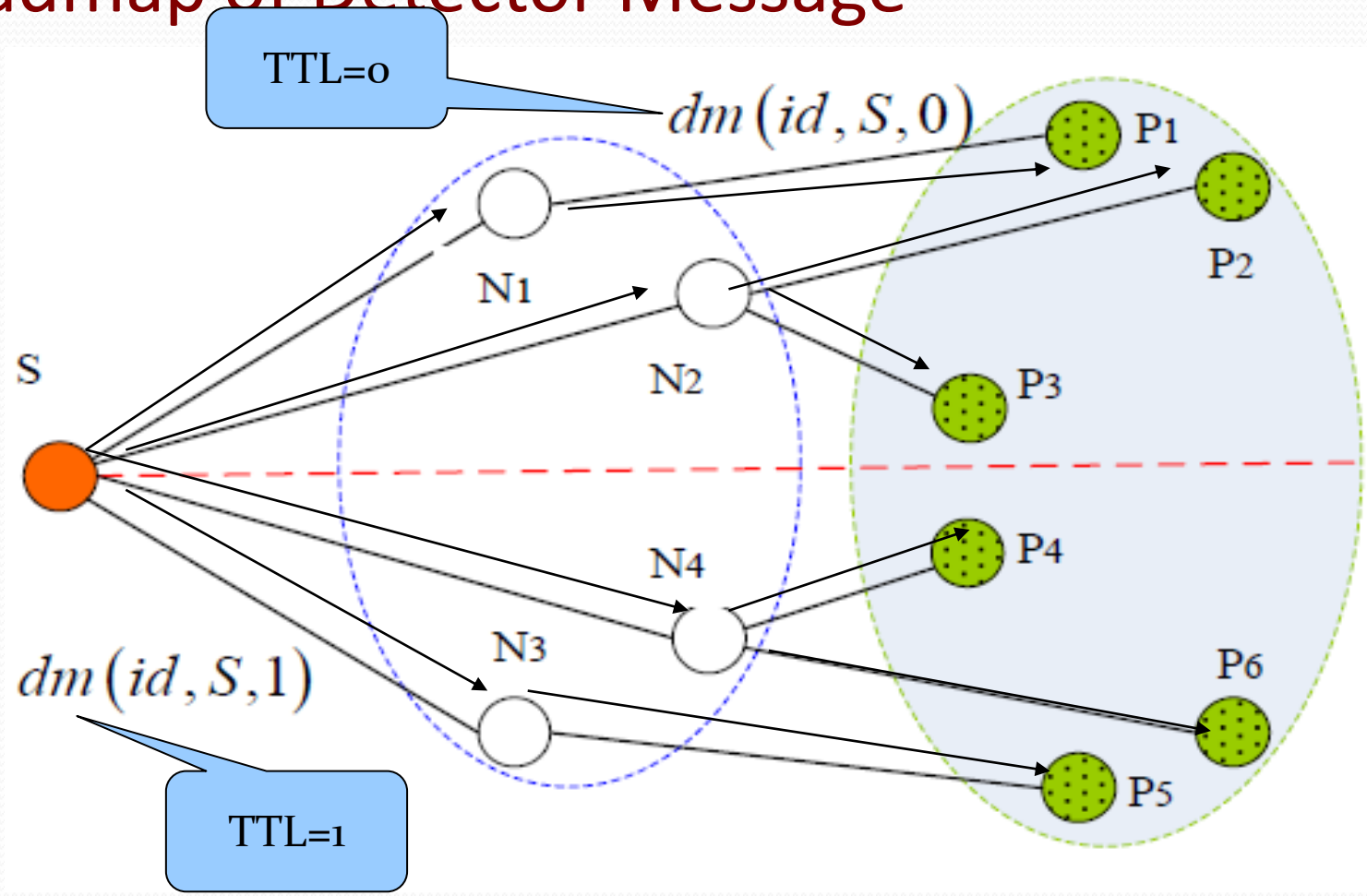
AnySee Design: Challenges

- How to find paths with low delays in a global P2P network
- How to maintain the service continuity and stability
- How to determine the frequency of optimization operations
- How to reduce the control overhead caused by the algorithm

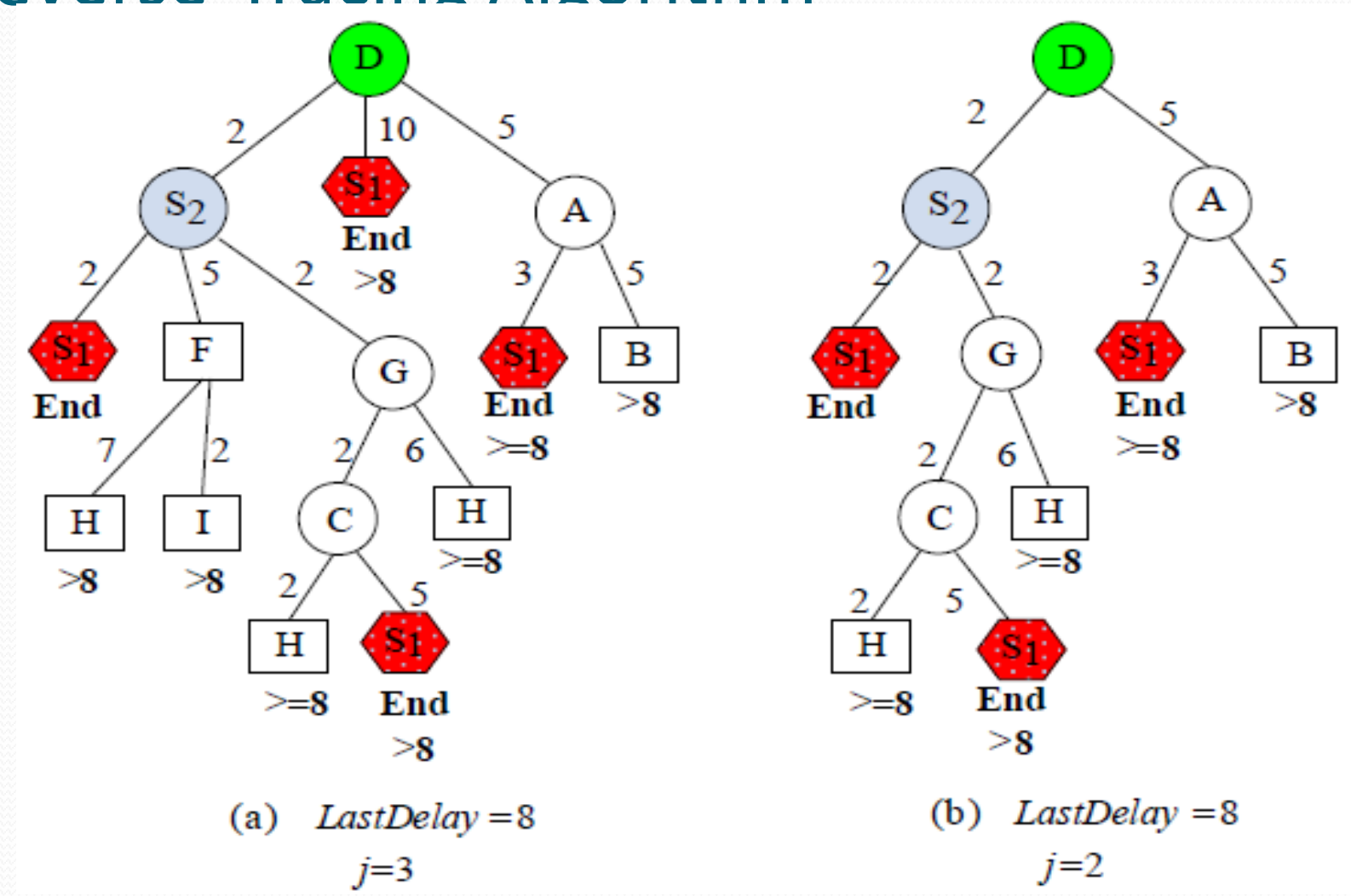
The System Diagram of an AnySee Node



Roadmap of Detector Message



Reverse-Tracing Algorithm



(a) $LastDelay = 8$, $j=3$

(b) $LastDelay = 8$, $j=2$

Inter-Overlay Optimization Manager

- Each peer maintains
  - one active streaming path set
  - one backup streaming path set

$$\sum_{i=1}^{\delta_a(P,S)} rate(SP_i, S, P) \geq rate(S) \quad (1)$$

$$\sum_{i=1}^{\delta_b(P,S)} rate(SP_i, S, P) = p \sum_{i=1}^{\delta_a(P,S)} rate(SP_i, S, P) \quad (2)$$

Characteristics of a Manager

- Employs a heuristic algorithm
  - The system is optimized step by step
- Probing procedures originate from the normal peers, not the source peer, so that the control overhead is balanced to normal peers
- The number of forwarding neighbors, *j*, balance the tradeoff between the optimization effectiveness and the overhead
- The frequency of probing and optimization is dynamic.

Queuing Model

Queuing Model (M/M/m/K)

$$p_n = \begin{cases} \dfrac{(m\rho)^n}{n!} p_0 & n = 0,1 \ldots m-1 \\ \dfrac{m^m \rho^n}{m!} p_0 & n = m, m+1 \ldots K \end{cases} \qquad (3)$$

$$p_0 = \begin{cases} \left[\sum_{i=0}^{m-1} \dfrac{(m\rho)^i}{i!} + \dfrac{(m\rho)^m}{m!} \dfrac{1-\rho^{K-m+1}}{1-\rho} \right]^{-1} & \rho \neq 1 \\ \left[\sum_{i=0}^{m-1} \dfrac{(m)^i}{i!} + \dfrac{(m)^m}{m!} (K-m+1) \right]^{-1} & \rho = 1 \end{cases} \qquad (4)$$

Optimizations

$$Max\left(\rho\left(N_1, N_2, \ldots N_M\right)\right) = Max\left(\sum_{1 \le i,j \le M}^{i \ne k}\left(\bar{\rho}_i + \tilde{\rho}_j\right)\right) \qquad (8)$$

$$Subject\ to \quad \sum_{i=1}^{M} N_i = N \quad 1 \le N_i < N$$

The above optimization problem can be divided into 2 parts

- we enumerate all (*M*, 1)-partitions of *N* spare connections
- For all *H* partitions of *N* connections, we can compute all *H* results of average resources utilization
  - select the best partition, based on which of the resources utilization is maximal.

$$H = \binom{N-1}{M-1} = \frac{(N-1)!}{(M-1)!(N-M)!},\ N \ge M \qquad (9)$$

Simulation Parameters

Abbreviate	Comment
S	Number of streaming overlays
M	Number of neighbors
N	Size of one overlay
r	Streaming playback rate
C	Number of total bandwidth connections

Results

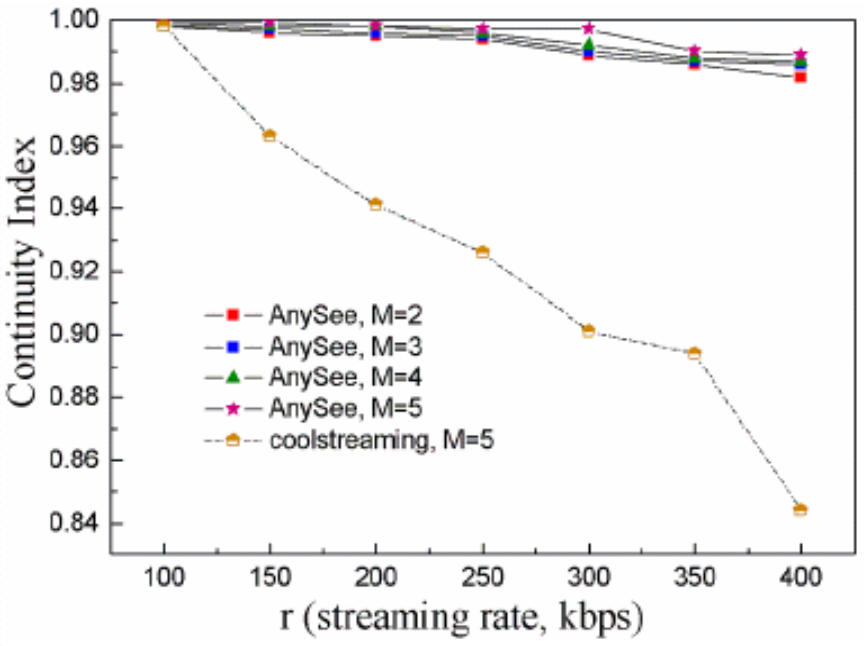


Figure: Continuity index V.S. streaming rates when N=400, S=12 and initial buffer size is 40 seconds [Figure 7 of the AnySee Paper]

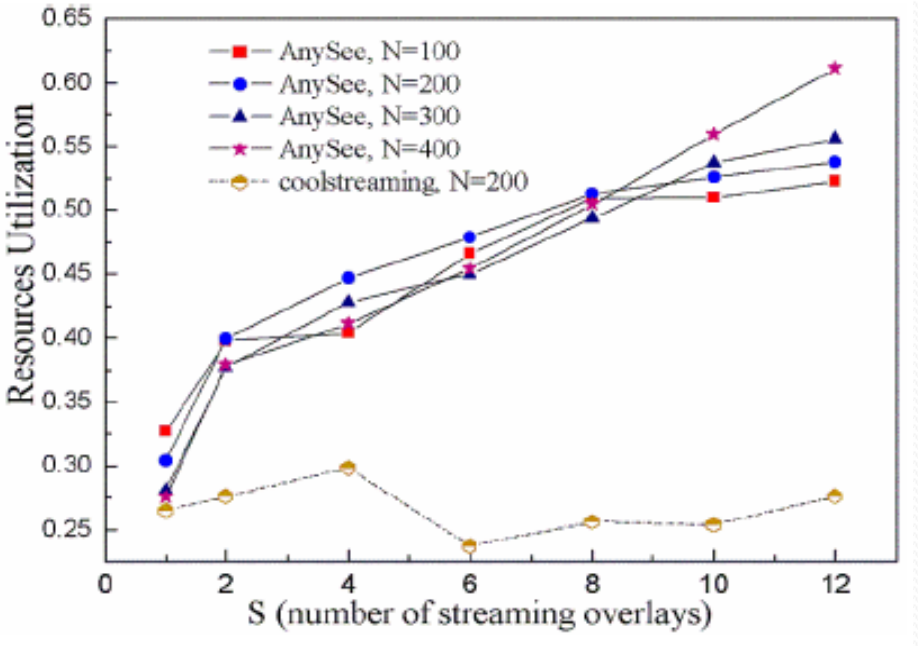


Figure: Resources utilization: overlay size V.S. the number of streaming overlays when M=12, r=300 Kbps [Figure 8 of the AnySee Paper]

Performance of AnySee

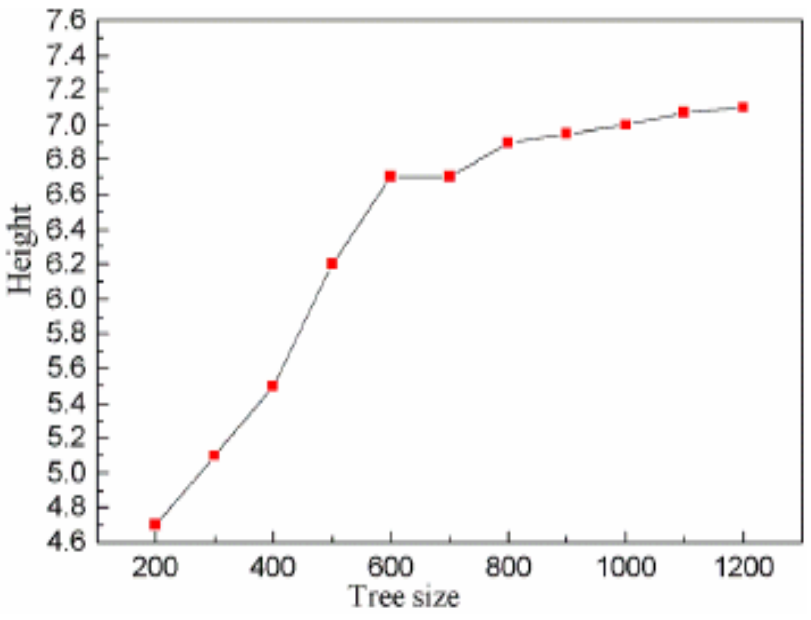


Figure: Height V.S. tree size [Figure 13 of AnySee Paper]

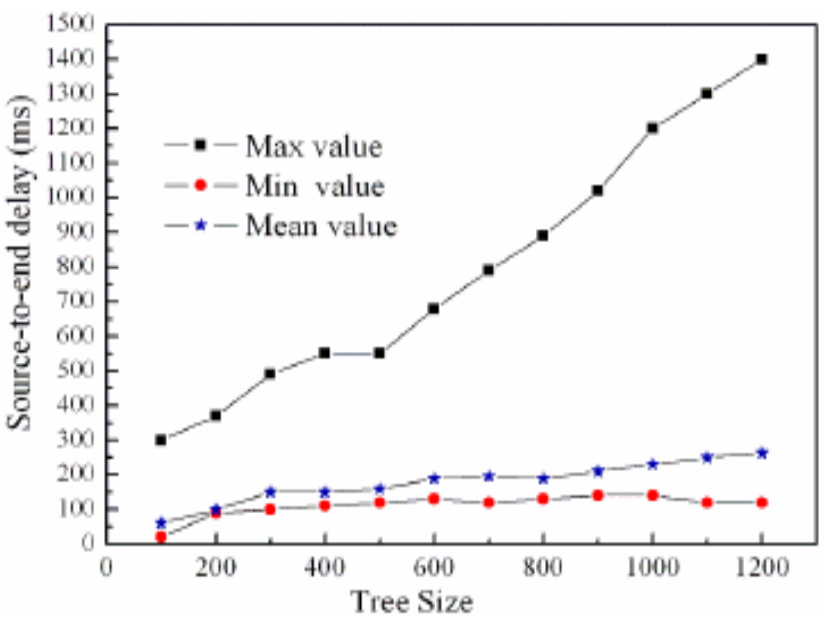


Figure: Source-to-end delay V.S. tree size [Figure 14 of AnySee Paper]

Discussions

- Better Global Resource Utilization
- Consider locality to minimize delay
- Better Load Balancing
- Scalable Approach
- Renamed as IOO Scheme

Thank You!

- Questions?

Beehive

- fully decentralized framework for resource allocation
- structured, self-organizing overlays (DHTs)
- An analysis-driven framework
  - to provide low-latency news dissemination
  - Limit the load placed on News Providers
    - Commercial Interests
    - Legal Bindings
  - Optimally trading off bandwidth for performance