Peer-to-Peer Media Streaming

Mohamed M. Hefeeda

Advisor: Prof. Bharat Bhargava

March 12, 2003

Outline

- Brief introduction to P2P
- Scope/Objective
- Current media streaming approaches
- Proposed approach: P2P framework
  - Definitions, P2P model
  - Advantages and challenges
- Architectures (realization of the model)
  - Hybrid
    - Searching and dispersion algorithms
  - Pure P2P (in progress)
- Evaluation
  - P2P model
  - Dispersion algorithm
- Conclusions and future work

P2P Systems: Basic Definitions

- Peers ***cooperate*** to achieve desired ***functions***
  - ***Cooperate:*** share resources (CPU, storage, bandwidth), participate in the protocols (routing, replication, …)
  - ***Functions:*** file-sharing, distributed computing, communications, …
- Examples
  - Gnutella, Napster, Freenet, OceanStore, CFS, CoopNet, SpreadIt, SETI@HOME, …
- Well, aren't they just distributed systems?
  - P2P == distributed systems?

P2P vs. Distributed Systems

- **P2P = distributed systems++;**
  - Ad-hoc nature
  - Peers are not servers [Saroui *et al.*, MMCN'02]
    - Limited capacity and reliability
  - Much more dynamism
  - Scalability is a more serious issue (millions of nodes)
  - Peers are self-interested (selfish!) entities
    - 70% of Gnutella users share nothing [Adar and Huberman '00]
  - All kind of Security concerns
    - Privacy, anonymity, malicious peers, … you name it!

P2P Systems: Rough Classification

[Lv *et al.*, ICS'02], [Yang *et al.*, ICDCS'02]

- Structured (or tightly controlled, DHT)
  - + Files are ***rigidly*** assigned to specific nodes
  - + Efficient search & guarantee of finding
  - – Lack of partial name and keyword queries
  - • Ex.: **Chord** [Stoica *et al.*, SIGCOMM'01], **CAN** [Ratnasamy *et al.*, SIGCOMM'01], **Pastry** [Rowstron and Druschel, Middleware'01]
- Unstructured (or loosely controlled)
  - + Files can be anywhere
  - + Support of partial name and keyword queries
  - – Inefficient search (some heuristics exist) & no guarantee of finding
  - • Ex.: Gnutella
- Hybrid (P2P + centralized), super peers notion)
  - \- Napster, KazaA

Scope/Objective

- A media streaming service (video on demand) that:
  - Provides good *quality*
  - To a *large* number of clients
  - In a *cost-effective* manner

- Main focus is on media *distribution* (or communication aspects)
- Media storage and encoding/decoding techniques are *orthogonal* to our work.

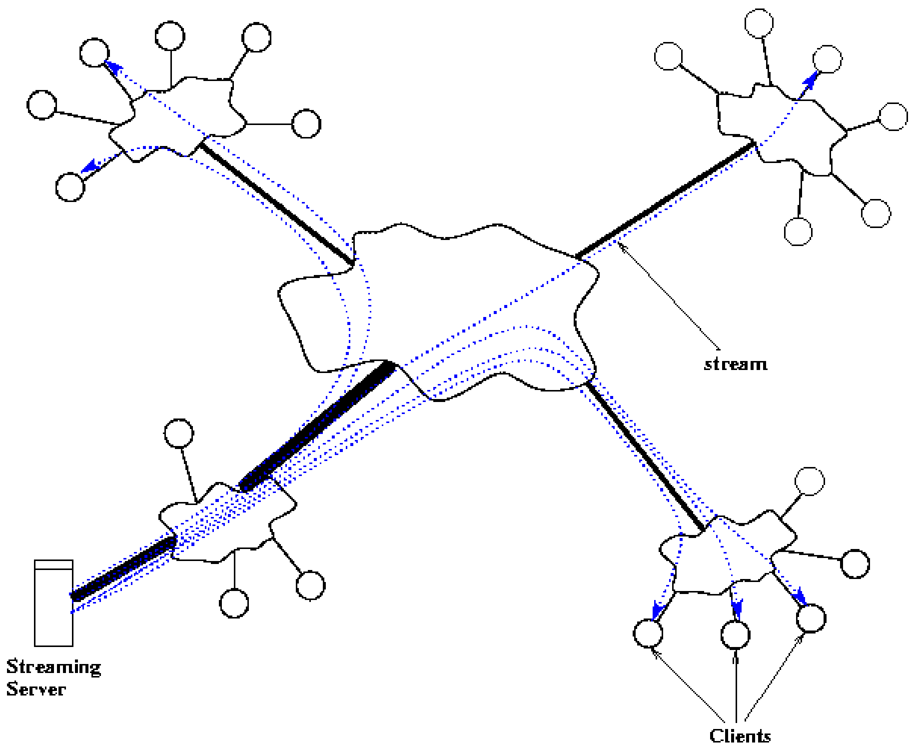
Classification of the Current Streaming Approaches

- Terminologies
  - Content provider
  - Clients
  - Third party (delivery)

- Two broad categories
  - Direct approach
    - Content provider ➔ clients
  - Third-party approach
    - Content provider ➔ delivery network ➔ clients

Direct Approach

- Content provider deploys and manages a powerful server or a set of servers/caches



Direct Approach (*cont'd*)

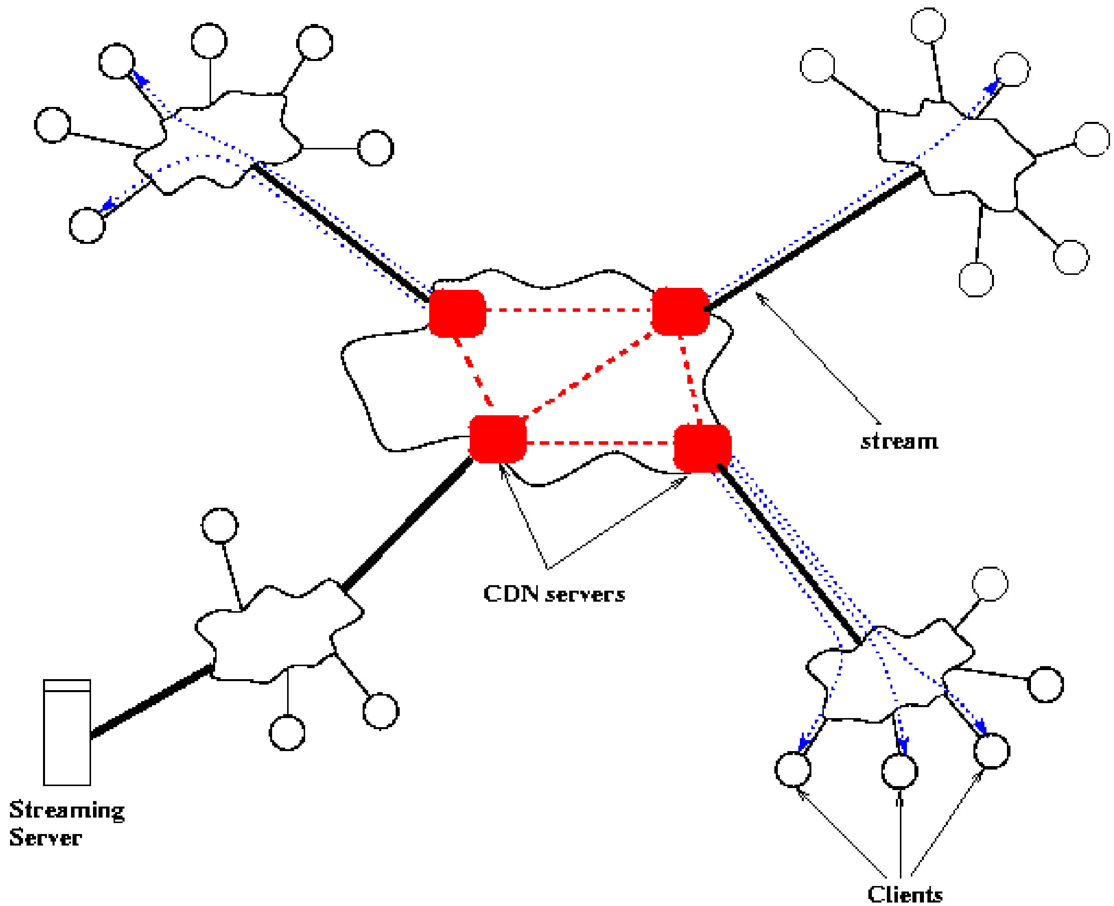
- Problems
  - Limited scalability
  - Reliability concerns
  - High deployment cost *$$$…..$*

- Note:
  - A server with T3 link (~45 Mb/s) supports up to 45 concurrent users at 1Mb/s!

Third-Party Approach

- Third-party or Content Delivery Network (CDN)
  - Deploy thousands of servers at the "edge" of the Internet; mainly at POPs of major ISPs (AT&T, Sprint, …)
    - (Akamai deploys 10,000+ servers) [Akamai white paper]
  - "Edge" of the Internet ➔
    - Contents close to clients
    - Better performance and less load on the backbone
  - Proprietary protocols to
    - Distribute contents over servers (caches)
    - Monitor traffic situation in the Internet
    - Direct clients to "most" suitable cache

Third-Party Approach (cont'd)



Third-Party Approach (cont'd)

- Pros
  - Good performance (short delay, more reliability, …)
  - Suitable for web pages with moderate-size objects (images, video clips, documents, etc.)
- Cons
  - Co$t: CDN charges for every megabyte served! ➔
  - Not suitable for VoD service; movies are quite large (~Gbytes)

- Note: [Raczkowski'02, white paper]
  - Cost ranges from 0.25 to 2 cents/MByte, depending on bandwidth consumed per month
  - For a one-hour movie streamed to 1,000 clients, content provider pays $264 to CDN (at 0.5 cents/MByte)!

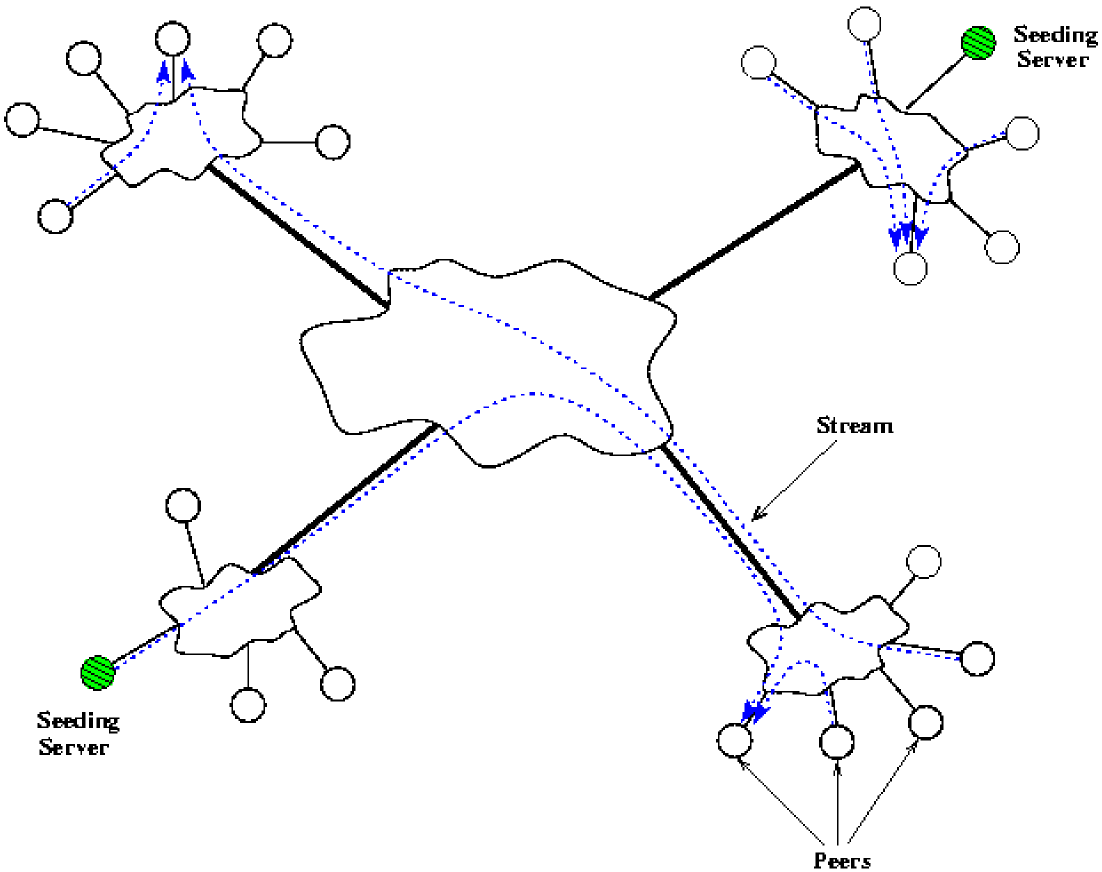
Potential Solution: P2P Model

- Idea
  - Clients (peers) *share* some of their spare resources (BW, storage) with each other
  - Result: combine enormous amount of resources into one pool ➔ significantly amplifies system capacity
  - Why should peers cooperate? [Saroui *et al.*, MMCN'02]
    - They get benefits too!
    - Incentives: e.g., lower rates
    - [Cost-profit analysis, Hefeeda *et al.*, TR'02]

P2P Model

Entities

- **Peers**
- **Seeding peers**
- **Stream**
- **Media files**



Proposed P2P model

P2P Model: Entities

- Peers
  - Supplying peers
    - Currently caching and willing to provide *some* segments
    - Level of cooperation; every peer P_x specifies:
      - G_x (Bytes),
      - R_x (Kb/s),
      - C_x (Concurrent connections)
  - Requesting peers
- Seeding peers
  - One (or a subset) of the peers *seeds* the new media into the system
  - Seed ≡ stream to a few other peers for a limited duration

P2P Model: Entities (*cont'd*)

- Stream
  - Time-ordered sequence of packets
- Media file
  - Recorded at ***R*** Kb/s (CBR)
  - Composed of ***N*** equal-length *segments*
  - A segment is the minimum unit to be cached by a peer
  - A segment can be obtained from several peers at the same time (different *piece* from each)

P2P Model: Advantages

- Cost effectiveness
  - For both supplier and clients
  - Initial results in [Hefeeda *et al.*, TR'02]
  - On-going work in cooperation with Professor Philipp Afeche (Kellogg School of Management, Northwestern University) to:
    - Develop more formal economic models
    - Design incentive schemes
    - Design pricing schemes
- Ease of deployment
  - No need to change the network (routers)
  - A piece of software on the client's machine

P2P Model: Advantages (*cont'd*)

- Robustness
  - High degree of redundancy
  - Reduce (gradually eliminate) the role of the seeding server

- Support for large number of clients
  - Capacity
    - More peers join ➔ more resources ➔ larger capacity
  - Network
    - Save downstream bandwidth; get the request from a nearby peer
    - Contents are even closer to the clients (within the same domain!)

P2P Model: Challenges

- Searching
  - Find peers who have the requested file
- Dispersion
  - Efficiently disseminate the media files into the system
- Maintaining comparable quality
  - Given a *dynamic* set of candidate senders, design a Distributed Streaming protocol that ensures the full quality of play back at the receiver
- Robustness
  - Handle node failures and network fluctuations
- Security
  - Malicious peers, free riders, …

Realization of the P2P Model

- Two architectures to realize the abstract model
- Hybrid [Hefeeda *et al.*, FTDCS'03; submitted to J. Com. Net.]
  - P2P streaming + index-assisted searching/dispersion
- Pure P2P
  - Peers form an *overlay* layer over the physical network
  - Built on top of a P2P substrate such as Pastry [Rowstron and Druschel, Middleware 2001]
  - On-going work

Hybrid Architecture

- Streaming is P2P; searching and dispersion are server-assisted
- Index server facilitates the searching process and reduces the overhead associated with it
- Suitable for a commercial service
  - Need server to charge/account anyway, and
  - Faster to deploy
- Seeding servers may maintain the index as well (especially, if commercial)

Hybrid Architecture: Searching

- Requesting peer, P_x
  - Send a request to the index server: <*fileID, IP, netMask*>
- Index server
  - Find peers who have segments of *fileID* AND *close* to P_x
  - *close* in terms of network hops ➔
    - Traffic traverses fewer hops, thus
    - Reduced load on the backbone
    - Less susceptible to congestion
    - Short and less variable delays (smaller delay jitter)
- Clustering idea [Krishnamurthy *et al.*, SIGCOMM'00]

Hybrid Architecture: Peers Clustering

- A cluster is:
  - A logical grouping of clients that are topologically close and likely to be within the same network domain
- Clustering Technique
  - Get routing tables from core BGP routers
  - Clients with IP's having the same longest prefix with one of the entries are assigned the same cluster ID
  - Example:
    - Domains: 128.10.0.0/16 (purdue), 128.2.0.0/16 (cmu)
    - Peers: 128.10.3.60, 128.10.3.100, 128.10.7.22, 128.2.10.1, 128.2.11.43

Hybrid Architecture: Dispersion

- Objective
  - Store *enough* copies of the media file in each cluster to serve all expected requests from that cluster
  - We assume that peers get *monetary* incentives from the provider to store and stream to other peers
- Questions
  - Should a peer cache? And if so,
  - Which segments?
- Illustration (media file with 2 segments)
  - Caching 90 copies of segment 1 and only 10 copies of segment 2 ➔ 10 effective copies
  - Caching 50 copies of segment 1 and 50 copies of segment 2 ➔ 50 effective copies

Hybrid Architecture: Dispersion (*cont'd*)

- Dispersion Algorithm (basic idea):
  - /* Upon getting a request from P_y to cache N_y segments */
  - $C \leftarrow$ *getCluster* (P_y)
  - Compute available (A) and required (D) capacities in cluster C
  - If $A < D$
    - P_y caches N_y segments in a *cluster-wide round robin* fashion (*CWRR)*

– All values are *smoothed* averages

– Average available capacity in C: $A_C = \frac{1}{T} \sum_{P_x \ in \ C} \frac{R_x}{R} \frac{N_x}{N} u_x$

– CWRR Example: (10-segment file)

- P_1 caches 4 segments: 1,2,3,4
- P_2 then caches 7 segments: 5,6,7,8,9,10,1

Hybrid Architecture: Client Protocol

- Building blocks of the protocol to be run by a *requesting* peer
- Three phases
  - Availability check
  - Streaming
  - Caching

Hybrid Architecture: Client Protocol (cont'd)

- Phase I: Availability check (who has what)
  - **Search** for peers that have segments of the requested file
  - **Arrange** the collected data into a 2-D table, row *j* contains all peers $\mathbf{P}^j$ willing to provide segment *j*
  - **Sort** every row based on ***network proximity***
  - **Verify** availability of all the ***N*** segments with the full rate ***R:***

$$\sum_{P_x \in \mathrm{P}^j} R_x \geq R$$

Hybrid Architecture: Client Protocol (*cont'd*)

- Phase II: Streaming

  $t_j = t_{j-1} + \delta$ /* δ: *time to stream a segment* */

  For $j = 1$ to N do

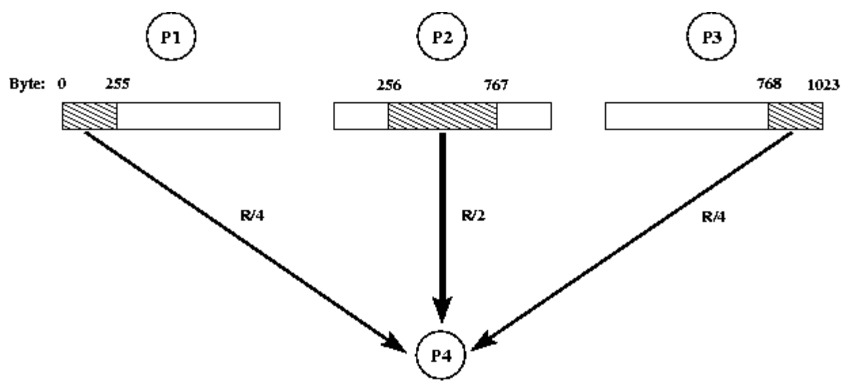
  At time t_j, get segment s_j as follows:

  - Connect to every peer $\boldsymbol{P_x}$ in $\mathbf{P}^j$ **(in parallel) and**
  - Download from byte $\boldsymbol{b_{x-1}}$ to $\boldsymbol{b_x - 1}$

Note: $\boldsymbol{b_x = |s_j|\, R_x / R}$

Example:

P_1, P_2, and P_3 serving different pieces of the same segment to P_4 with different rates



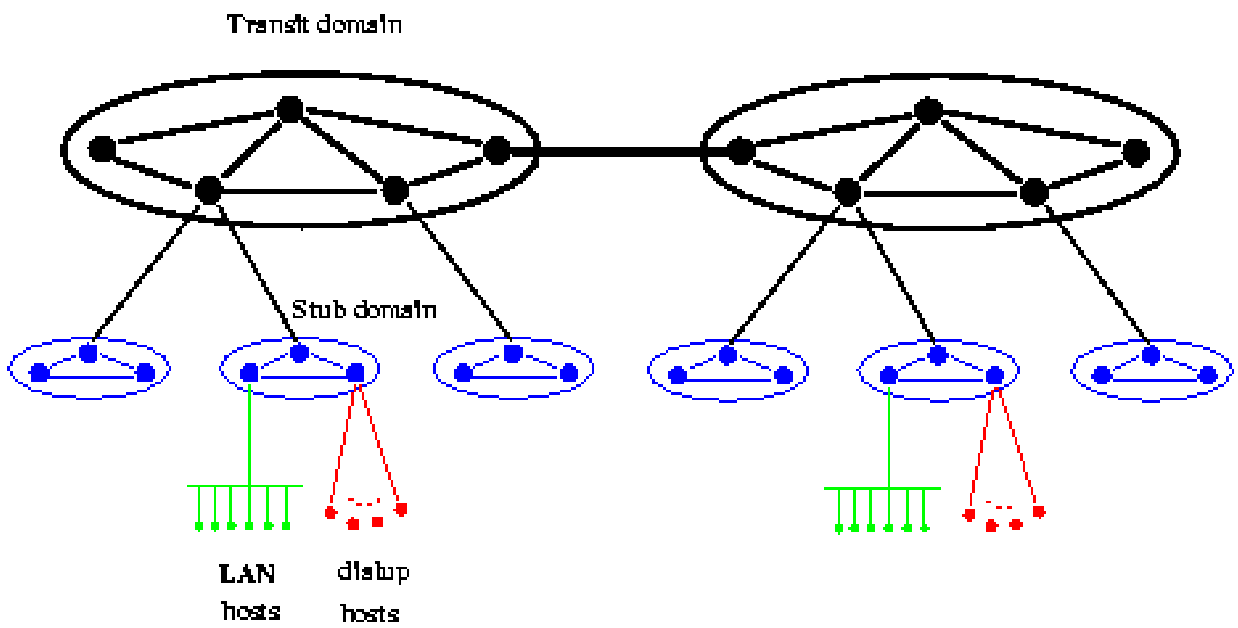
Hybrid Architecture: Client Protocol (*cont'd*)

- Phase III: Caching
  - Store some segments
  - Determined by the *dispersion* algorithm, and
  - Peer’s level of cooperation

Evaluation Through Simulation

- Performance of the hybrid architecture
  - Under several client arrival patterns (constant rate, flash crowd, Poisson) and different levels of peer cooperation
  - Performance measures
    - Overall system capacity,
    - Average waiting time,
    - Average number of served (rejected) requests, and
    - Load/Role on the seeding server
- Performance of the dispersion algorithm
  - Compare against random dispersion algorithm

Simulation: Topology



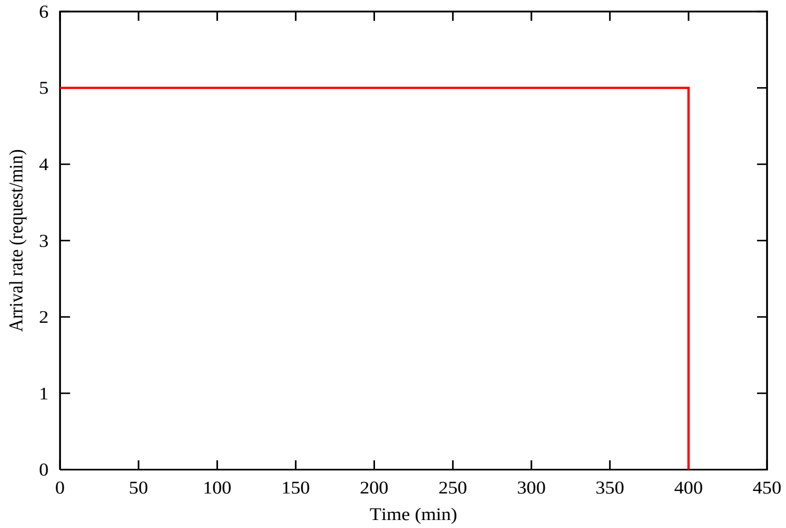
– Large (more than 13,000 nodes)
– Hierarchical (Internet-like)
– Used *GT-ITM* and *ns-2*

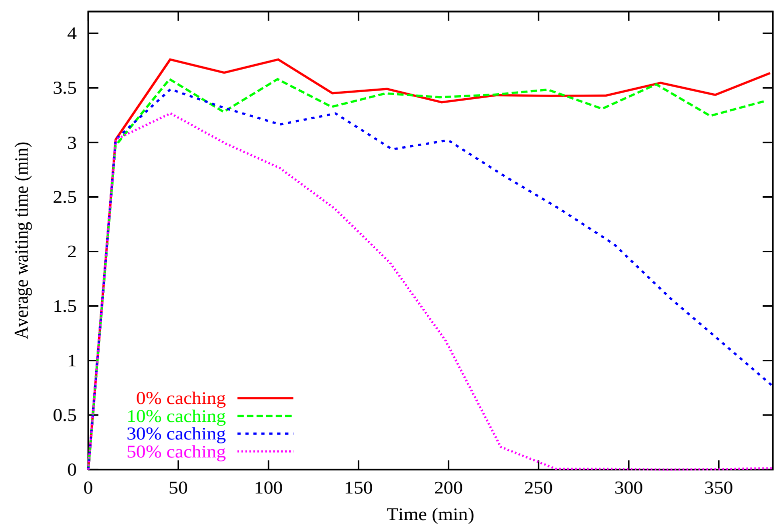
Hybrid Architecture Evaluation

- Topology details
  - 20 transit domains, 200 stub domains, 2,100 routers, and a total of 11,052 end hosts
- Scenario
  - A seeding server with limited capacity (up to 15 clients) introduces a movie
  - Clients request the movie according to the simulated arrival pattern
  - Client protocol is applied
- Fixed parameters
  - Media file of 20 min duration, divided into 20 one-min segments, and recorded at 100 Kb/s (CBR)

Hybrid Architecture Evaluation (*cont'd*)

- Constant rate arrivals: *waiting time*

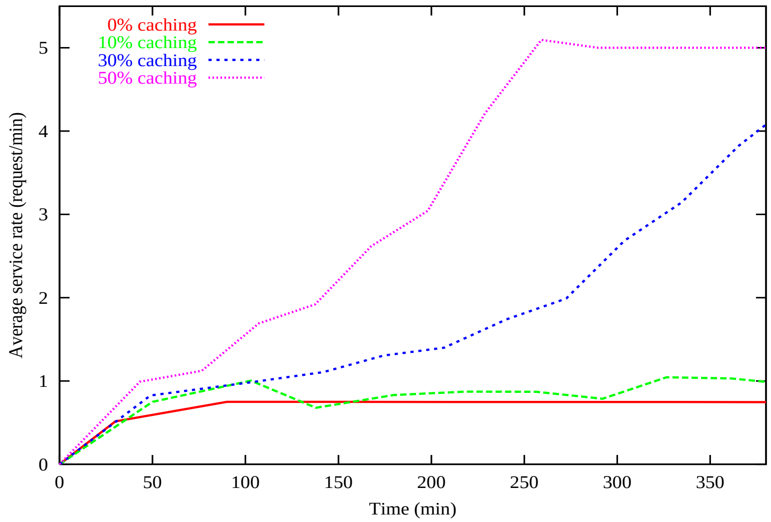


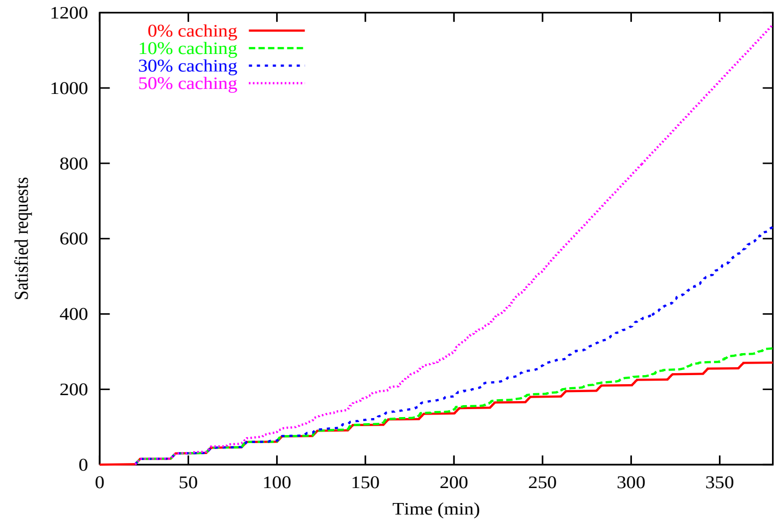


– Average waiting time decreases as the time passes
  - It decreases faster with higher caching percentages

Hybrid Architecture: Evaluation (*cont'd*)

- Constant rate arrivals: *service rate*

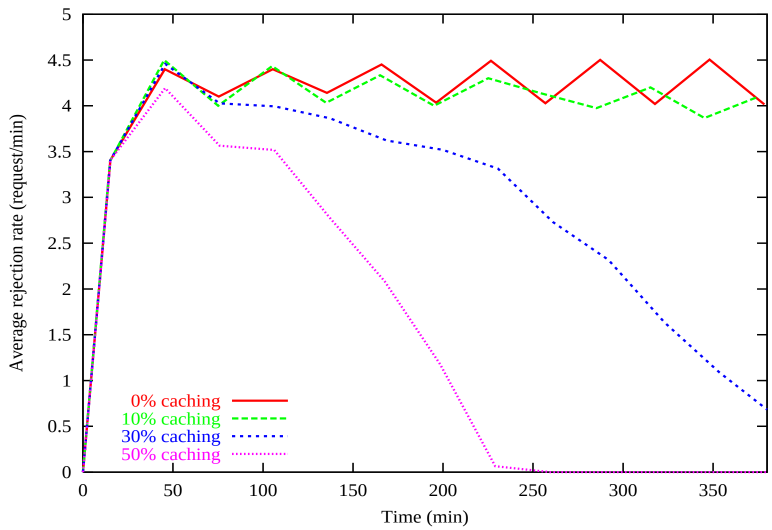


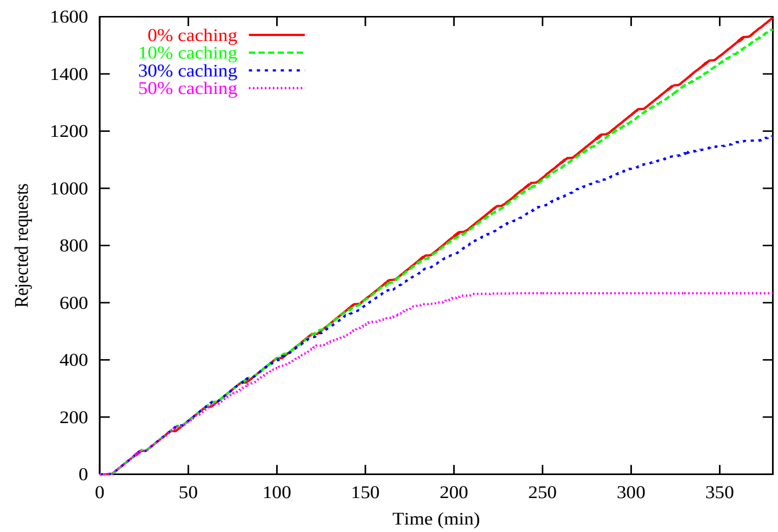


- Capacity is rapidly amplified
  - All requests are satisfied after 250 minutes with 50% caching
- Q: Given a target arrival rate, what is the appropriate caching%? When is the steady state?
  - Ex.: 2 req/min ➔ 30% sufficient, steady state within 5 hours

Hybrid Architecture: Evaluation (*cont'd*)

- Constant rate arrivals: *rejection rate*

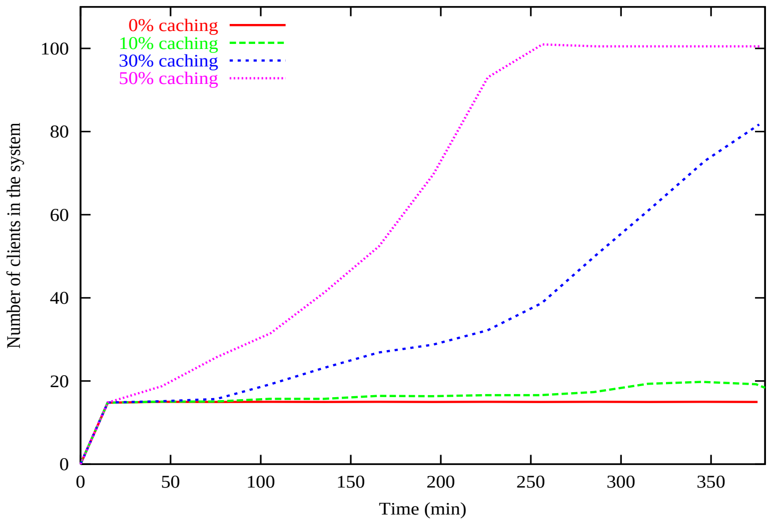


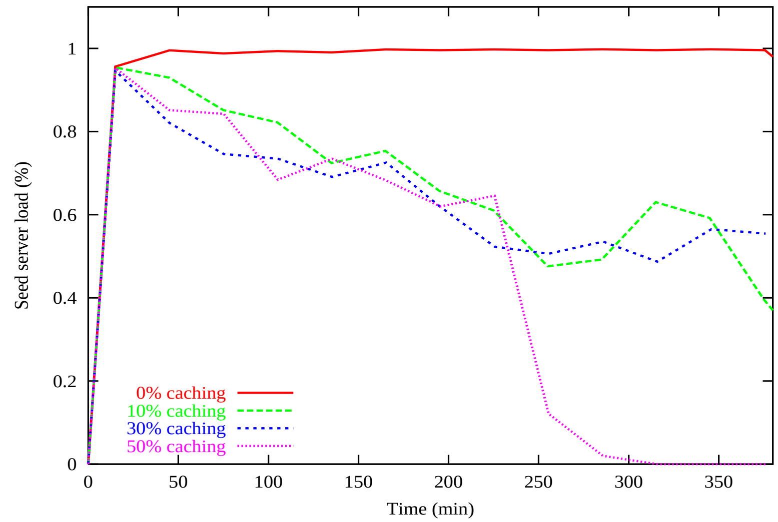


– Rejection rate is decreasing with time
  - No rejections after 250 minutes with 50% caching
– Longer *warm up* period is needed for smaller caching percentages

Hybrid Architecture: Evaluation (*cont'd*)

- Constant rate arrivals: *load on the seeding server*

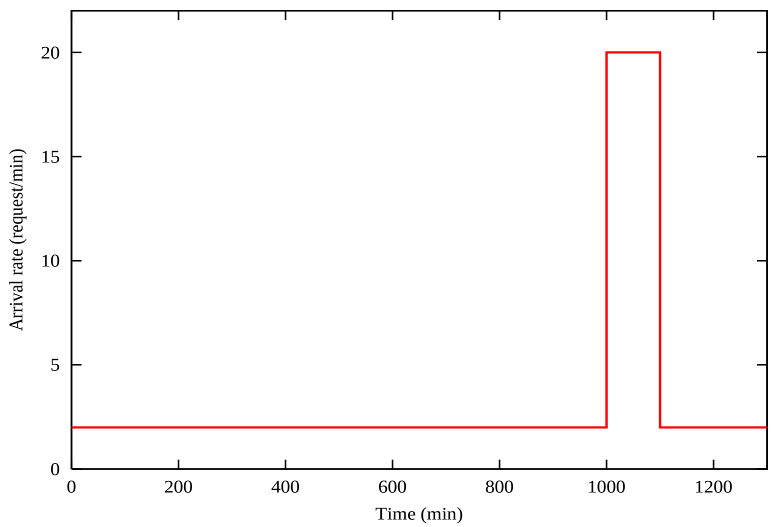


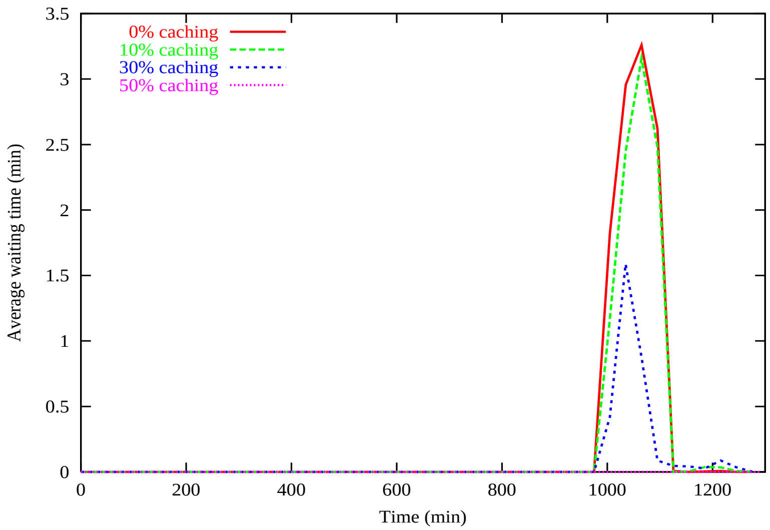


  – The role of the seeding server is *diminishing*
    - For 50%: After 5 hours, we have *100* concurrent clients (6.7 times original capacity) and *none* of them is served by the seeding server

Hybrid Architecture: Evaluation (*cont'd*)

- Flash crowd arrivals: *waiting time*

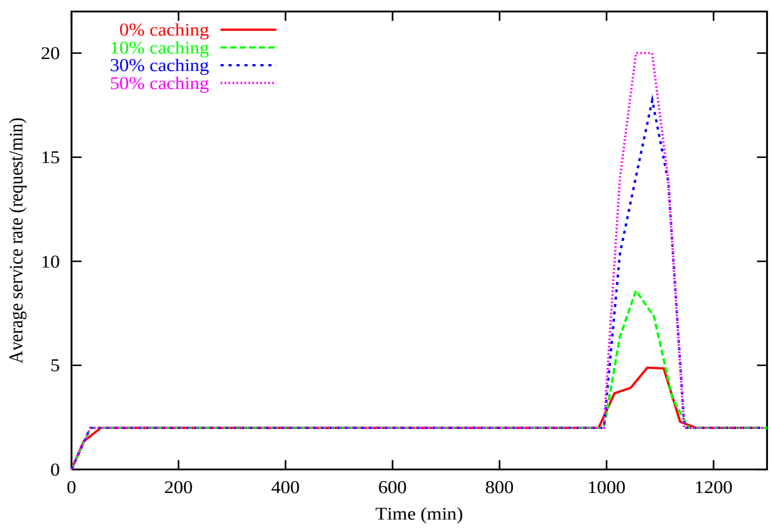


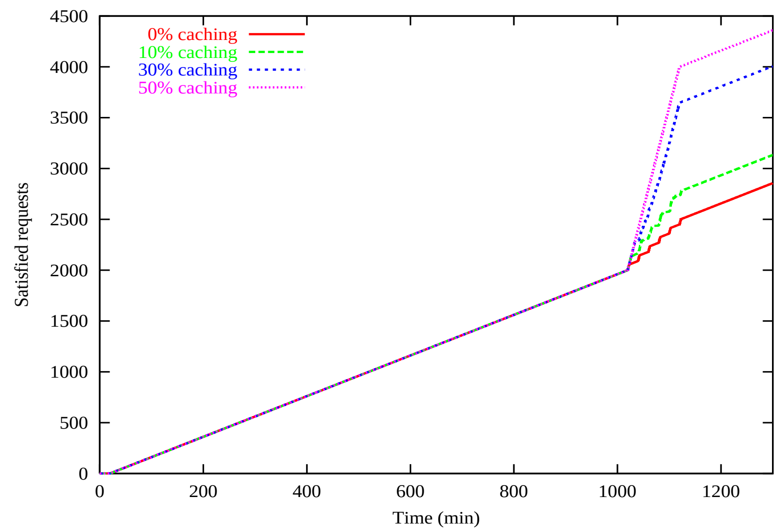


  - Flash crowd arrivals ≡ surge increase in client arrivals
  - Waiting time is zero even during the peak (with 50% caching)

Hybrid Architecture: Evaluation (*cont'd*)

- Flash crowd arrivals: *service rate*

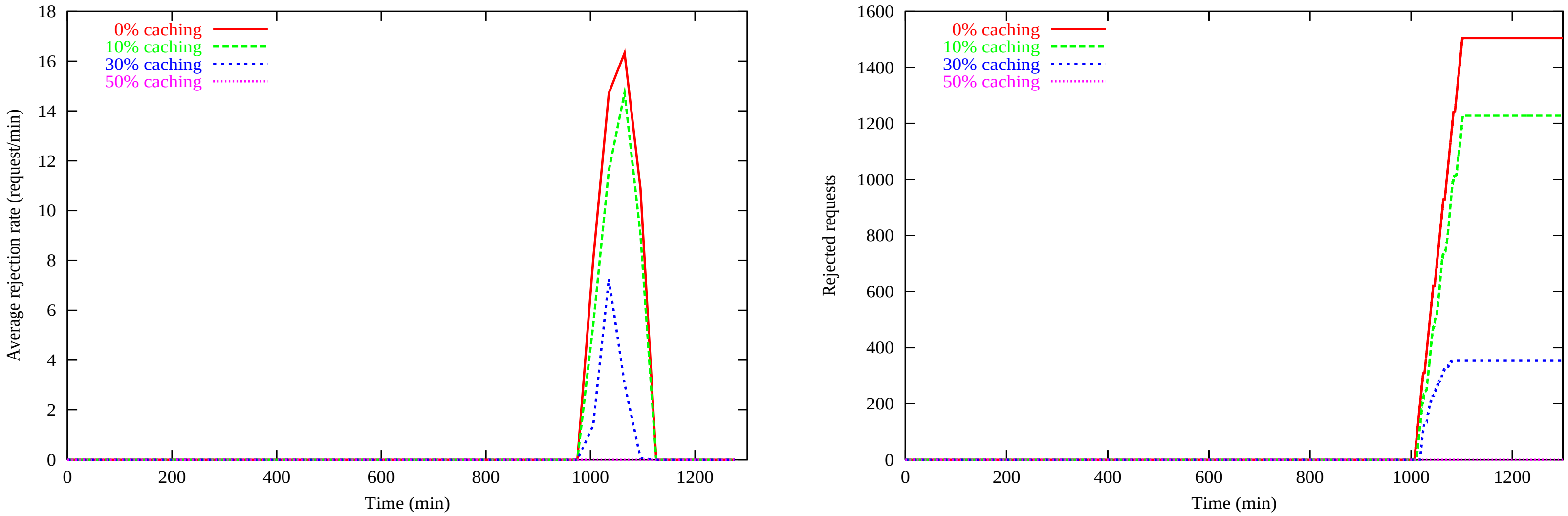




- All clients are served with 50% caching
- Smaller caching percentages need longer warm up periods to fully handle the crowd

Hybrid Architecture: Evaluation (*cont'd*)

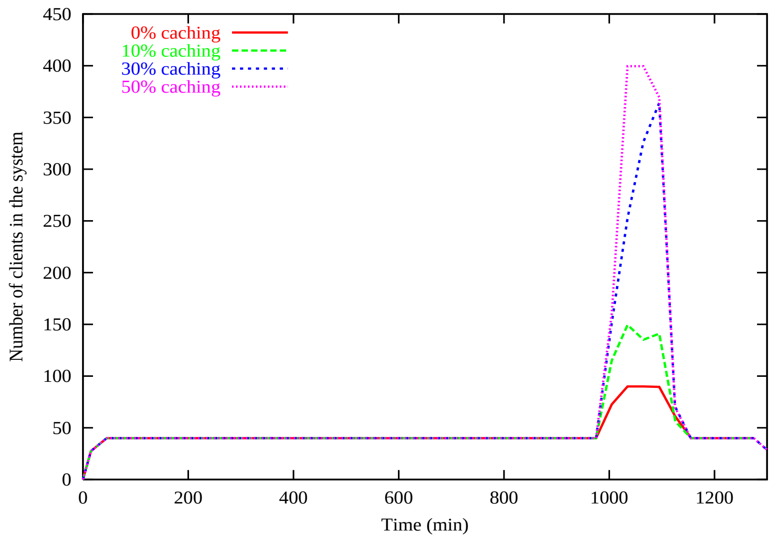
- Flash crowd arrivals: *rejection rate*

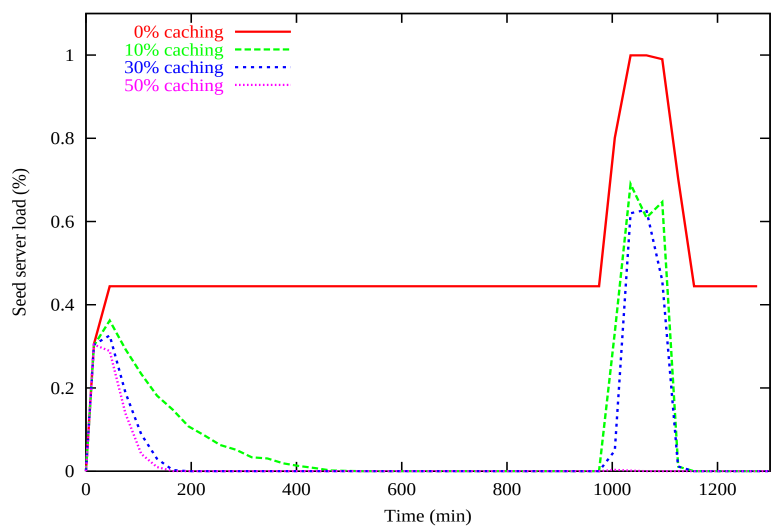


  – No clients turned away with 50% caching

Hybrid Architecture: Evaluation (*cont'd*)

- Flash crowd arrivals: *load on the seeding server*





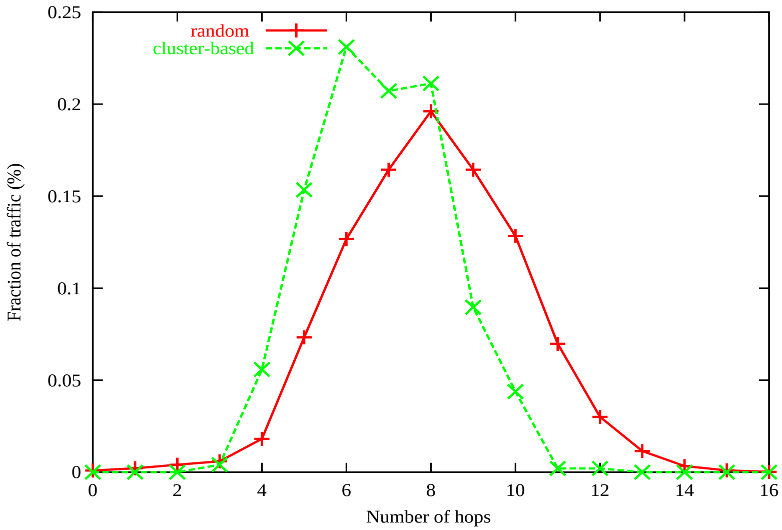
– The role of the seeding server is still just *seeding*
  - During the peak, we have *400* concurrent clients (26.7 times original capacity) and *none* of them is served by the seeding server (50% caching)

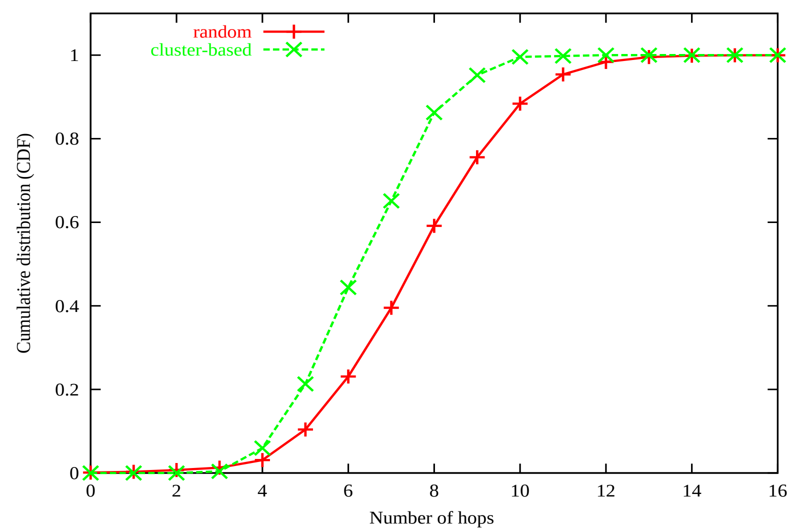
Dispersion Algorithm: Evaluation

- Topology details
  - 100 transit domains, 400 stub domains, 2,400 routes, and a total of 12,021 end hosts
    - Distribute clients over a ***wider*** range ➔ more stress on the dispersion algorithm
- Compare against a *random* dispersion algorithm
  - No other dispersion algorithms fit our model
- Comparison criterion
  - Average number of network hops traversed by the stream
- Vary the caching percentage from 5% to 90%
  - Smaller cache % ➔ more stress on the algorithm

Dispersion Algorithm: Evaluation (*cont'd*)

5% caching

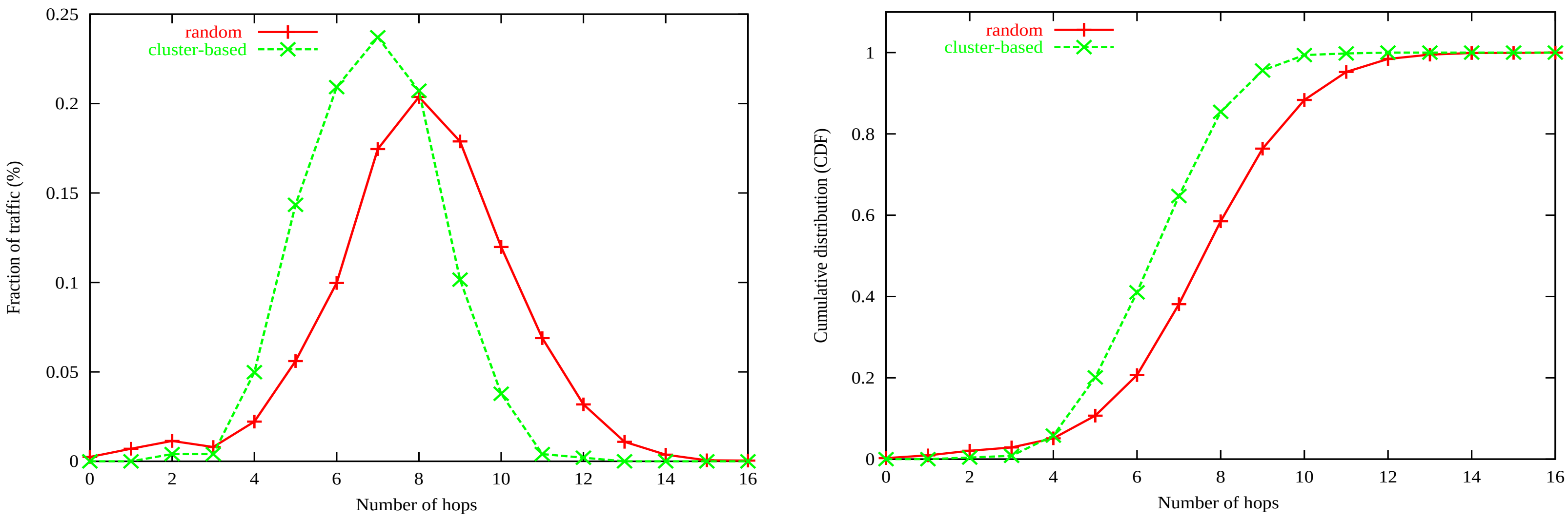




– Avg. number of hops:
  - 8.05 hops (random), 6.82 hops (ours) ➔ 15.3% savings
– For a domain with a 6-hop diameter:
  - Random: 23% of the traffic was kept inside the domain
  - Cluster-based: 44% of the traffic was kept inside the domain

Dispersion Algorithm: Evaluation (*cont'd*)

10% caching



– As the caching percentage increases, the difference decreases; peers cache most of the segments, hence no room for enhancement by the dispersion algorithm

Conclusions

- Presented a new model for on-demand media streaming
- Proposed two architectures to realize the model
  - Hybrid and Pure P2P
- Presented dispersion and searching algorithms
- Through large-scale simulation, we showed that
  - Our model successfully supports large number of clients
    - Arriving to the system with various distributions, including flash crowds
  - Our dispersion algorithm pushes the contents close to the clients (within the same domain) ➔
    - Reduces number of hops traversed by the stream and the load on the network

Future Work

- Work out the details of the overlay approach
- Address the reliability and security challenges
- Develop a detailed cost-profit model for the P2P architecture to show its cost effectiveness compared to the conventional approaches
- Implement a system prototype and study other performance metrics, e.g., delay, delay jitter, and loss rate
- Enhance the proposed algorithms and *formally* analyze them

P2P: File-sharing vs. Streaming

- File-sharing
  - Download the *entire* file first, then use it
  - Small files (few Mbytes) ➔ short download time
  - A file is stored by one peer ➔ one connection
  - No timing constraints
- Streaming
  - Consume (playback) as you download
  - Large files (few Gbytes) ➔ long download time
  - A file is stored by multiple peers ➔ several connections
  - Timing is crucial

Current Streaming Approaches (*cont'd*)

- P2P approaches
  - SpreadIt [Deshpande *et al.*, Stanford TR'01]
    - Live media
      - Build application-level multicast distribution tree over peers
  - CoopNet [Padmanabhan *et al.*, NOSSDAV'02 and IPTPS'02]
    - Live media
      - Builds application-level multicast distribution tree over peers
    - On-demand
      - Server redirects clients to other peers
      - Assumes a peer can (or is willing to) support the full rate
      - CoopNet does not address the issue of quickly disseminating the media file

Current Streaming Approaches (*cont'd*)

- Distributed caches [e.g., Chen and Tobagi, ToN'01]
  - Deploy caches all over the place
  - Yes, increases the scalability
    - Shifts the bottleneck from the server to caches!
  - But, it also multiplies cost
  - What to cache? And where to put caches?
- Multicast
  - Mainly for live media broadcast
  - Application level [Narada, NICE, Scattercast, …]
    - Efficient?
  - IP level [e.g., Dutta and Schulzrine, ICC'01]
    - Widely deployed?