

Peer-to-peer Video Systems. Structure and challenges

Alberto Montresor Gianluca Ciccarelli

Networking group - University of Trento

April 30, 2009

(ciccarelli@disi.unitn.it)

Outline

- 1 Terminology
- 2 Overlay management
 - PeerCast (structured)
 - System description
 - Experiments and results
 - End-system behaviour: model
 - End-System Multicast (unstructured)
 - CoolStreaming (data-driven)
- 3 General measurement results

Outline

- 1 Terminology
- 2 Overlay management
 - PeerCast (structured)
 - System description
 - Experiments and results
 - End-system behaviour: model
 - End-System Multicast (unstructured)
 - CoolStreaming (data-driven)
- 3 General measurement results

Outline

- 1 Terminology
- 2 Overlay management
 - PeerCast (structured)
 - System description
 - Experiments and results
 - End-system behaviour: model
 - End-System Multicast (unstructured)
 - CoolStreaming (data-driven)
- 3 General measurement results

Video compression

- To make its transmission more convenient, a video stream is usually compressed using a lossy compression.
- Compression algorithms:
 - Discrete Cosine Transform (DCT)
 - Vector Quantization (VQ)
 - Fractal compression
 - Discrete Wavelet Transform (DWT)
- Some widespread compression standards:
 - ITU-T's H.26x (e.g., H.264) rates of multiples of 64 Kbit/s, DCT-based
 - ISO/IEC's MPEG-x (e.g., MPEG-4 AVC), MPEG-1 supports a rate up to 1.5 Mbit/s, MPEG-2 up to 15 Mbit/s
 - VPx (e.g., VP8), used by Adobe Flash

Video compression

- To make its transmission more convenient, a video stream is usually compressed using a lossy compression.
- Compression algorithms:
 - Discrete Cosine Transform (DCT)
 - Vector Quantization (VQ)
 - Fractal compression
 - Discrete Wavelet Transform (DWT)
- Some widespread compression standards:
 - ITU-T's H.26x (e.g., H.264) rates of multiples of 64 Kbit/s, DCT-based
 - ISO/IEC's MPEG-x (e.g., MPEG-4 AVC), MPEG-1 supports a rate up to 1.5 Mbit/s, MPEG-2 up to 15 Mbit/s
 - VPx (e.g., VP8), used by Adobe Flash

Video compression

- To make its transmission more convenient, a video stream is usually compressed using a lossy compression.
- Compression algorithms:
 - Discrete Cosine Transform (DCT)
 - Vector Quantization (VQ)
 - Fractal compression
 - Discrete Wavelet Transform (DWT)
- Some widespread compression standards:
 - ITU-T's H.26x (e.g., H.264) rates of multiples of 64 Kbit/s, DCT-based
 - ISO/IEC's MPEG-x (e.g., MPEG-4 AVC), MPEG-1 supports a rate up to 1.5 Mbit/s, MPEG-2 up to 15 Mbit/s
 - VPx (e.g., VP8), used by Adobe Flash

Compression principles

- Human eye sees no difference when some information is taken out of the video
- The lossy compression algorithms are based on the loss of some information to obtain an acceptable video quality at feasible encoding/decoding rates
- Lossless algorithms are an expensive alternative

Motion prediction

- Motion prediction is used to improve the speed of the encoding/decoding process
- Idea: Encode a frame f , then predict the structure of a new frame using f
- Drawback: Errors in the transmission of a frame make every subsequent frame impossible to reconstruct
- Solution: Occasionally encode one video frame using still-image coding techniques only ("intra frames" or "I-frames")
 - P-frames: Frames encoded using only a previously displayed reference frame
 - B-frames: encoded using both future and previously displayed reference frames

Motion prediction

- Motion prediction is used to improve the speed of the encoding/decoding process
- Idea: Encode a frame f , then predict the structure of a new frame using f
- Drawback: Errors in the transmission of a frame make every subsequent frame impossible to reconstruct
- Solution: Occasionally encode one video frame using still-image coding techniques only ("intra frames" or "I-frames")
 - P-frames: Frames encoded using only a previously displayed reference frame
 - B-frames: encoded using both future and previously displayed reference frames

Motion prediction

- Motion prediction is used to improve the speed of the encoding/decoding process
- Idea: Encode a frame f , then predict the structure of a new frame using f
- Drawback: Errors in the transmission of a frame make every subsequent frame impossible to reconstruct
- Solution: Occasionally encode one video frame using still-image coding techniques only ("intra frames" or "I-frames")
 - P-frames: Frames encoded using only a previously displayed reference frame
 - B-frames: encoded using both future and previously displayed reference frames

Example

- Workings: the codec encodes an I-frame, skips several frames ahead and encodes a future P-frame using the encoded I-frame as a reference frame, then skips back to the next frame following the I-frame. The frames between the encoded I- and P-frames are encoded as B-frames. This process continues, with a new I-frame inserted for every 12 to 15 P- and B-frames.

		Display Order (left to right)											
Encoding Order (top to bottom)	I												
					P								
		B											
			B										
				B									
								P					
					B								
						B							
							B					P	
								B					
									B				
										B			
												I	

Requirements for video

- General requirements: no jitter (variable delay), no artifacts
- Video-on-Demand is the transmission of pre-recorded video content to a requesting user
 - The content can be replicated and delivered from the closest server
 - Set-top boxes solutions store content at the user end-point
- Live streaming is the transmission of content which is being created in real time
 - Low-latency requirement: content is created and streamed almost at the same time
 - A few seconds of delay may be unacceptable (soccer matches)
 - No possibility to strategically pre-replicate the content
 - P2PTV: what about zapping?
- Live streaming is the main challenge in P2P video systems

Overlays

- An overlay is a virtual network maintained at the application level
- The control can be centralized or distributed
- A P2P overlay is widely used for the distribution of content
 - The control can still be centralized, like in Skype

Overlays

- An overlay is a virtual network maintained at the application level
- The control can be centralized or distributed
- A P2P overlay is widely used for the distribution of content
 - The control can still be centralized, like in Skype

Design of Overlays

- An overlay can be built in different ways:
 - Application Layer Multicast (ALM)
 - Structured (overlay management, typically tree-structured, DHT-based) vs. Unstructured (no explicit management, typically mesh)
 - Structured examples: NICE, ZigZag, SpreadIT
 - Unstructured examples: ESM, Narada
 - Data-driven overlay
 - The stream is divided into chunks which are disseminated by the peers
 - Examples: Coolstreaming, PPLive, SplitStream

Design of Overlays

- An overlay can be built in different ways:
 - Application Layer Multicast (ALM)
 - Structured (overlay management, typically tree-structured, DHT-based) vs. Unstructured (no explicit management, typically mesh)
 - Structured examples: NICE, ZigZag, SpreadIT
 - Unstructured examples: ESM, Narada
 - Data-driven overlay
 - The stream is divided into chunks which are disseminated by the peers
 - Examples: Coolstreaming, PPLive, SplitStream

Design of Overlays

- An overlay can be built in different ways:
 - Application Layer Multicast (ALM)
 - Structured (overlay management, typically tree-structured, DHT-based) vs. Unstructured (no explicit management, typically mesh)
 - Structured examples: NICE, ZigZag, SpreadIT
 - Unstructured examples: ESM, Narada
 - Data-driven overlay
 - The stream is divided into chunks which are disseminated by the peers
 - Examples: Coolstreaming, PPLive, SplitStream

Selecting nodes and chunks

- How to select nodes? how to select chunks?
- Selection of chunks:
 - Buffer map
 - Push protocol: the node decides to whom sending data
 - Pull protocol: the node requires a chunk from another
- Selection of nodes:
 - Random selection
 - Neighbourhood
 - Algorithm selection (e.g., FlightPath)
 - FlightPath: each node has an ID
 - A node i selects a node j if the hash result of the concatenation $HASH(ID(i), ID(j))$ is less than a threshold p

Selecting nodes and chunks

- How to select nodes? how to select chunks?
- Selection of chunks:
 - Buffer map
 - Push protocol: the node decides to whom sending data
 - Pull protocol: the node requires a chunk from another
- Selection of nodes:
 - Random selection
 - Neighbourhood
 - Algorithm selection (e.g., FlightPath)
 - FlightPath: each node has an ID
 - A node i selects a node j if the hash result of the concatenation $HASH(ID(i), ID(j))$ is less than a threshold p

Selecting nodes and chunks

- How to select nodes? how to select chunks?
- Selection of chunks:
 - Buffer map
 - Push protocol: the node decides to whom sending data
 - Pull protocol: the node requires a chunk from another
- Selection of nodes:
 - Random selection
 - Neighbourhood
 - Algorithm selection (e.g., FlightPath)
 - FlightPath: each node has an ID
 - A node i selects a node j if the hash result of the concatenation $HASH(ID(i), ID(j))$ is less than a threshold p

PeerCast: introduction

- Example of **structured** overlay for video distribution
- The model of the architecture:
 - defines the **data** exchanged between nodes (stream)
 - identify the roles of the **nodes** (clients, servers)
 - provides some details about the **communication** mechanism (peering layer, sessions)

PeerCast: conceptual model

- Stream: sequence of ordered pkts
- Stream = data channel (RTP/UDP) + control channel (RTSP/TCP)
- RTP = Real-time Transport Protocol (defines pkt format),
RTSP = Real Time Streaming Protocol (control of streaming servers)
- Each stream is identified through a URL
- *Live stream* \Rightarrow *history-agnostic*

PeerCast: conceptual model

- Stream: sequence of ordered pkts
- Stream = data channel (RTP/UDP) + control channel (RTSP/TCP)
- RTP = Real-time Transport Protocol (defines pkt format),
RTSP = Real Time Streaming Protocol (control of streaming servers)
- Each stream is identified through a URL
- *Live stream* \Rightarrow *history-agnostic*

PeerCast: conceptual model

- Stream: sequence of ordered pkts
- Stream = data channel (RTP/UDP) + control channel (RTSP/TCP)
- RTP = Real-time Transport Protocol (defines pkt format),
RTSP = Real Time Streaming Protocol (control of streaming servers)
- Each stream is identified through a URL
- *Live stream* \Rightarrow *history-agnostic*

PeerCast: conceptual model

- Stream: sequence of ordered pkts
- Stream = data channel (RTP/UDP) + control channel (RTSP/TCP)
- RTP = Real-time Transport Protocol (defines pkt format),
RTSP = Real Time Streaming Protocol (control of streaming servers)
- Each stream is identified through a URL
- *Live stream* \Rightarrow *history-agnostic*

PeerCast: nodes (I)

- Source s :
 - Stays up and running for the entire streaming session
 - Encodes information in the URL of each stream
 - Has a well-known address
- A client:
 - Sends a **subscribe** request to the source
 - Is directed by the source to choose a server
 - Sends an **unsubscribe** request when is no longer interested in the transmission

PeerCast: nodes (I)

- Source s :
 - Stays up and running for the entire streaming session
 - Encodes information in the URL of each stream
 - Has a well-known address
- A client:
 - Sends a **subscribe** request to the source
 - Is directed by the source to choose a server
 - Sends an **unsubscribe** request when is no longer interested in the transmission

PeerCast: nodes (II)

- The set of clients \mathcal{N} changes over time as subscriptions change
- Nodes are uniquely identified through their IP address
- A node that cannot serve a client with acceptable QoS is said to be *saturated*, *unsaturated* otherwise
- A node waiting for a stream (after subscription) is said to be in *transient state*
- Nodes are organized into groups
- Each group is organized according to a spanning tree rooted at the source and maintained as nodes come and go

PeerCast: nodes (II)

- The set of clients \mathcal{N} changes over time as subscriptions change
- Nodes are uniquely identified through their IP address
- A node that cannot serve a client with acceptable QoS is said to be *saturated*, *unsaturated* otherwise
- A node waiting for a stream (after subscription) is said to be in *transient state*
- Nodes are organized into groups
- Each group is organized according to a spanning tree rooted at the source and maintained as nodes come and go

PeerCast: nodes (II)

- The set of clients \mathcal{N} changes over time as subscriptions change
- Nodes are uniquely identified through their IP address
- A node that cannot serve a client with acceptable QoS is said to be *saturated*, *unsaturated* otherwise
- A node waiting for a stream (after subscription) is said to be in *transient state*
- Nodes are organized into groups
- Each group is organized according to a spanning tree rooted at the source and maintained as nodes come and go

PeerCast: nodes (II)

- The set of clients \mathcal{N} changes over time as subscriptions change
- Nodes are uniquely identified through their IP address
- A node that cannot serve a client with acceptable QoS is said to be *saturated*, *unsaturated* otherwise
- A node waiting for a stream (after subscription) is said to be in *transient state*
- Nodes are organized into groups
- Each group is organized according to a spanning tree rooted at the source and maintained as nodes come and go

PeerCast: communication (I)

- **Data-transfer session**: instance of data + control channel between a client and a server
- **Application session**: Interval between subscription and unsubscription
 - Comprises many data-transfer sessions
 - How many?
- The two session concepts are conceptually separated, but many unicast applications bind them together

PeerCast: communication (I)

- **Data-transfer session**: instance of data + control channel between a client and a server
- **Application session**: Interval between subscription and unsubscription
 - Comprises many data-transfer sessions
 - How many?
- The two session concepts are conceptually separated, but many unicast applications bind them together

PeerCast: communication (I)

- **Data-transfer session**: instance of data + control channel between a client and a server
- **Application session**: Interval between subscription and unsubscription
 - Comprises many data-transfer sessions
 - How many?
- The two session concepts are conceptually separated, but many unicast applications bind them together

PeerCast: communication (I)

- **Data-transfer session**: instance of data + control channel between a client and a server
- **Application session**: Interval between subscription and unsubscription
 - Comprises many data-transfer sessions
 - How many? Equal to the number of servers that a client switches between subscription and unsubscription
- The two session concepts are conceptually separated, but many unicast applications bind them together

PeerCast: communication (II)

- Peering layer between transport and application layer to decouple the sessions
 - Data-transfer sessions are established between the peering layers of the two end-points
 - Application sessions are established in each end-point between the peering layer and the application layer
 - The peering layer manages the server connection establishment, the application layer only uses a *getStream* interface

Peering layer: primitives (II)

- Discover:
 - a newly joined peer n contacts the server s at the well-known URL
 - if s is unsaturated, n becomes its child
 - otherwise, s redirects n to one of its children and the process is repeated (iteratively)
- Unsubscribe:
 - If peer c decides to unsubscribe, it sends a message to its parent p
 - c also redirects all its children to another unsaturated node (s or p)

Peering layer: primitives (II)

- Discover:
 - a newly joined peer n contacts the server s at the well-known URL
 - if s is unsaturated, n becomes its child
 - otherwise, s redirects n to one of its children and the process is repeated (iteratively)
- Unsubscribe:
 - If peer c decides to unsubscribe, it sends a message to its parent p
 - c also redirects all its children to another unsaturated node (s or p)

Topology management

Lemma

Under homogeneous unicast edge and node characteristics, an almost-complete spanning tree is the optimal overlay tree for packet loss, packet delay, and time to first packet metrics

- Authors compare their model with a hypothetical optimal system which (according to the lemma) maintains an almost-complete tree
- The hypothesis of homogeneity of node characteristics has become more and more unrealistic over time!

Topology management: Join policies

- Peer n receives a request of join from peer c , but is saturated
- How does n redirects c (given that n only knows about its local topology)?
 - 1 **Random**: n chooses one of its children. On average, the tree is balanced
 - 2 **Round-Robin (RR)**: n keeps a list of the children. Every time n is saturated, redirects c to the child that is the head of the list t , and then puts t at the end of the list. The resulting tree is expected to be balanced
 - 3 **Smart-Placement (SP)**: Each peer knows where in the network each of its children is, and redirects c to the child that has the minimum access latency from c
 - What does c need to send?

Topology management: Join policies

- Peer n receives a request of join from peer c , but is saturated
- How does n redirects c (given that n only knows about its local topology)?
 - 1 **Random**: n chooses one of its children. On average, the tree is balanced
 - 2 **Round-Robin** (RR): n keeps a list of the children. Every time n is saturated, redirects c to the child that is the head of the list t , and then puts t at the end of the list. The resulting tree is expected to be balanced
 - 3 **Smart-Placement** (SP): Each peer knows where in the network each of its children is, and redirects c to the child that has the minimum access latency from c
 - What does c need to send?

Topology management: Join policies

- Peer n receives a request of join from peer c , but is saturated
- How does n redirects c (given that n only knows about its local topology)?
 - 1 **Random**: n chooses one of its children. On average, the tree is balanced
 - 2 **Round-Robin (RR)**: n keeps a list of the children. Every time n is saturated, redirects c to the child that is the head of the list t , and then puts t at the end of the list. The resulting tree is expected to be balanced
 - 3 **Smart-Placement (SP)**: Each peer knows where in the network each of its children is, and redirects c to the child that has the minimum access latency from c
 - What does c need to send?

Topology management: Join policies

- Peer n receives a request of join from peer c , but is saturated
- How does n redirects c (given that n only knows about its local topology)?
 - 1 **Random**: n chooses one of its children. On average, the tree is balanced
 - 2 **Round-Robin (RR)**: n keeps a list of the children. Every time n is saturated, redirects c to the child that is the head of the list t , and then puts t at the end of the list. The resulting tree is expected to be balanced
 - 3 **Smart-Placement (SP)**: Each peer knows where in the network each of its children is, and redirects c to the child that has the minimum access latency from c
 - What does c need to send?

Topology management: Join policies

- Peer n receives a request of join from peer c , but is saturated
- How does n redirects c (given that n only knows about its local topology)?
 - 1 **Random**: n chooses one of its children. On average, the tree is balanced
 - 2 **Round-Robin** (RR): n keeps a list of the children. Every time n is saturated, redirects c to the child that is the head of the list t , and then puts t at the end of the list. The resulting tree is expected to be balanced
 - 3 **Smart-Placement** (SP): Each peer knows where in the network each of its children is, and redirects c to the child that has the minimum access latency from c
 - **What does c need to send?** Its traceroute information (to help n locate it)

Topology management: peer deletion

- Node n unsubscribes
- Two alternatives can be indicated by n : its parent and the source
 - \Rightarrow all its *descendants* go in transient state
 - ...but when the direct children C recover feed, their descendants recover automatically
- Policies for selection of an alternative:
 - 1 **Root-All** (RTA): choose s . All the descendants contact s . The tree remains balanced (redistribution)
 - 2 **Grandfather-All** (GFA): choose the parent, p . All the descendants contact p . Only n 's subtree is rebuilt, which remains balanced
 - 3 **Root** (RT): choose s . Only the children C contact s . The effects of failures are localized
 - 4 **Grandfather** (GF): choose p . Only the children contact p . The effects of failures are localized

Topology management: peer deletion

- Node n unsubscribes
- Two alternatives can be indicated by n : its parent and the source
 - \Rightarrow all its *descendants* go in transient state
 - ...but when the direct children C recover feed, their descendants recover automatically
- Policies for selection of an alternative:
 - 1 **Root-All** (RTA): choose s . All the descendants contact s . The tree remains balanced (redistribution)
 - 2 **Grandfather-All** (GFA): choose the parent, p . All the descendants contact p . Only n 's subtree is rebuilt, which remains balanced
 - 3 **Root** (RT): choose s . Only the children C contact s . The effects of failures are localized
 - 4 **Grandfather** (GF): choose p . Only the children contact p . The effects of failures are localized

Topology management: peer deletion

- Node n unsubscribes
- Two alternatives can be indicated by n : its parent and the source
 - \Rightarrow all its *descendants* go in transient state
 - ...but when the direct children C recover feed, their descendants recover automatically
- Policies for selection of an alternative:
 - 1 **Root-All** (RTA): choose s . All the descendants contact s . The tree remains balanced (redistribution)
 - 2 **Grandfather-All** (GFA): choose the parent, p . All the descendants contact p . Only n 's subtree is rebuilt, which remains balanced
 - 3 **Root** (RT): choose s . Only the children C contact s . The effects of failures are localized
 - 4 **Grandfather** (GF): choose p . Only the children contact p . The effects of failures are localized

Overlay improvements

- Overlay improvements are based on heuristics
- Nodes x and y compute a metric $F(x, y)$ according to network proximity, unicast latency, and bandwidth capacity
- Any node n computes periodically $F(n, c)$ for all his children

Overlay improvements

- Overlay improvements are based on heuristics
- Nodes x and y compute a metric $F(x, y)$ according to network proximity, unicast latency, and bandwidth capacity
- Any node n computes periodically $F(n, c)$ for all his children

Overlay improvements

- Overlay improvements are based on heuristics
- Nodes x and y compute a metric $F(x, y)$ according to network proximity, unicast latency, and bandwidth capacity
- Any node n computes periodically $F(n, c)$ for all his children

Example: Join procedure

- A peer x asks n to join

```
// n computes  $F(n, x)$ 
joinCost  $\leftarrow F(n, x)$ ;
if (n.isUnsaturated()) {
    n.accept(x);
} else {
    if ( $\exists c' \in C: \text{joinCost} < F(n, c)$ ) {
        n.discard(c');
        n.accept(x);
        n.redirect(c');
    } else {
        n.reject(x);
    }
}
```

- Other policies exist, based on the same or on similar metrics

Roadmap for evaluation

- On-the-field experiments and measurements to obtain the values of key parameters
- The parameters are then used in simulation to investigate the peer's behaviour

Experimental settings

- Source: Apple's Darwin streaming server
- Peering layer written in Python
- Join policy: Random
- Leave policy: Root (RT)
- Nodes failure occur when heartbeat stops
- Metrics (informal):
 - RTP/UDP pkt **delay**
 - Time to first pkt (TTFP)
 - RTP/UDP pkt **drops** (lost blocks rather than lost pkts)

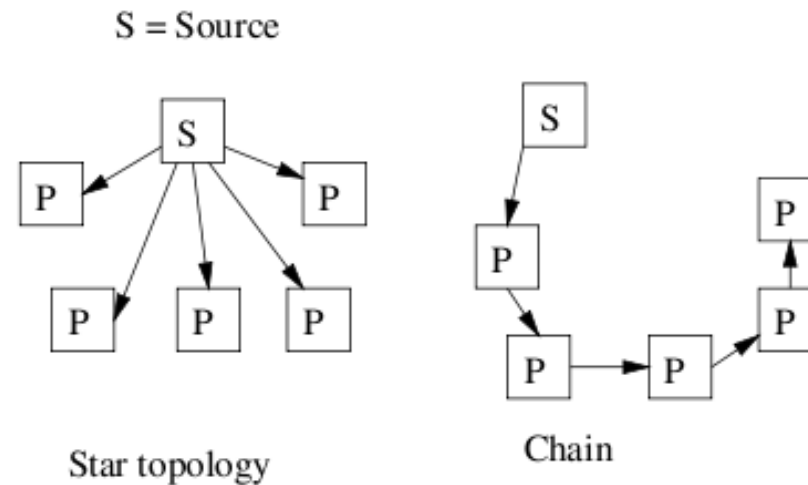
Experimental settings

- Source: Apple's Darwin streaming server
- Peering layer written in Python
- Join policy: Random
- Leave policy: Root (RT)
- Nodes failure occur when heartbeat stops
- Metrics (informal):
 - RTP/UDP pkt **delay**
 - Time to first pkt (TTFP)
 - RTP/UDP pkt **drops** (lost blocks rather than lost pkts)

Experimental settings

- Source: Apple's Darwin streaming server
- Peering layer written in Python
- Join policy: Random
- Leave policy: Root (RT)
- Nodes failure occur when heartbeat stops
- Metrics (informal):
 - RTP/UDP pkt **delay**
 - Time to first pkt (TTFP)
 - RTP/UDP pkt **drops** (lost blocks rather than lost pkts)

Topologies



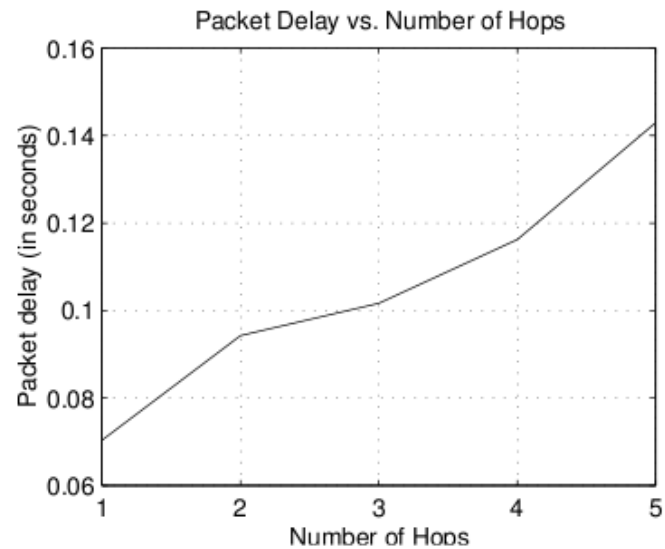
Notes: Topologies used in the experiments

- The chain topology is used to measure the performance when the number of hops increases
- Two sets of experiments: all nodes in the same intranet, and all nodes but the source in the same intranet

Definition of metrics

- Metrics (formal):
 - RTP/UDP pkt delays: $t_C - t_S$ (t_S : time the source sends the pkt; T_C : time the client receives it)
 - TTFP: $t_1 - t_J$ (t_J : time the new node joins the system; t_1 : time it receives the first pkt)
 - RTP/UDP pkt drops (effective only when consecutive thanks to the redundant encoding)

Delays



Notes: Chain topology (depth 5), clients and source in the same intranet

- Order of magnitude (5 hops): 0.14s; buffer size: 30s
- For a chain of 15 hops (3×5) we have
 $3 \times 0.14 = 0.42 \approx 1.5\%$ with respect to the 30s of the buffer

Stream latency and Packet drops

- TTFP = Time to discover an unsaturated node (also known as *Stream latency*): linear in the number of hops, 2 orders of magnitude smaller than the node buffer (30s)
 - The *redirect* primitive can be considered efficient
- Packet losses: Observations suggest to support children in the same intranet (avg **half** pkt losses with respect to losses when the source is across the Internet)

Stream latency and Packet drops

- TTFP = Time to discover an unsaturated node (also known as *Stream latency*): linear in the number of hops, 2 orders of magnitude smaller than the node buffer (30s)
 - The *redirect* primitive can be considered efficient
- Packet losses: Observations suggest to support children in the same intranet (avg **half** pkt losses with respect to losses when the source is across the Internet)

How observations impact design

- Minimize jitter \rightarrow large buffer (30s)
- Only a small fraction of it assigned to control (join, leave, redirect): say 5% \rightarrow 1.5s
- Observation: connection establishment over the Internet takes 1.3s
 - \Rightarrow 0.2s for redirect
 - 1 redirect takes circa 0.01s \Rightarrow 20 redirects possible
- This implies we can manage trees with height 20 (number of redirect operations allowed)

Model for simulations

- Source s sends out pkts every t_s steps to N homogeneous clients
- The stream lasts t_{len} steps
- Pkt loss probability p_{loss}
- Sending a pkt takes 1 simulation step (discrete-event simulation)
- Each client supports up to c_{max} children and manages an event queue, that leads the simulation (nodes respond to events). Three different events:
 - DESCRIBE (request connection), Duration: t_c
 - SETUP (request hand-shake), t_{hs}
 - PLAY (request start of the stream), t_{ss}

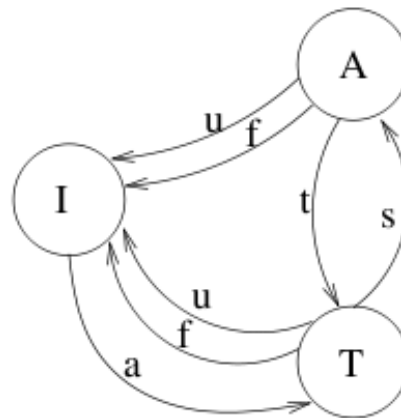
Model for simulations

- Source s sends out pkts every t_s steps to N homogeneous clients
- The stream lasts t_{len} steps
- Pkt loss probability p_{loss}
- Sending a pkt takes 1 simulation step (discrete-event simulation)
- Each client supports up to c_{max} children and manages an event queue, that leads the simulation (nodes respond to events). Three different events:
 - DESCRIBE (request connection), Duration: t_c
 - SETUP (request hand-shake), t_{hs}
 - PLAY (request start of the stream), t_{ss}

Model for simulations

- Source s sends out pkts every t_s steps to N homogeneous clients
- The stream lasts t_{len} steps
- Pkt loss probability p_{loss}
- Sending a pkt takes 1 simulation step (discrete-event simulation)
- Each client supports up to c_{max} children and manages an event queue, that leads the simulation (nodes respond to events). Three different events:
 - DESCRIBE (request connection), Duration: t_c
 - SETUP (request hand-shake), t_{hs}
 - PLAY (request start of the stream), t_{ss}

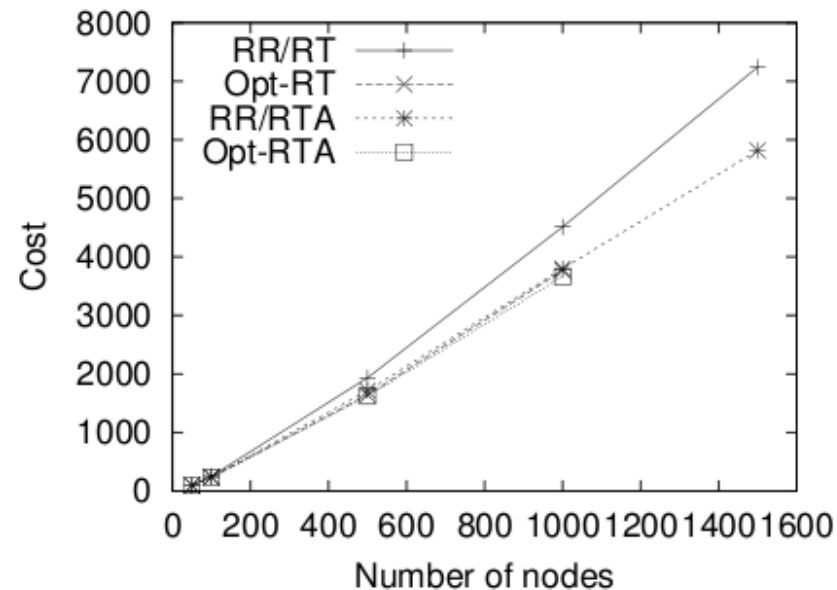
States of a node



Notes: I: inactive, A: active, T: transient; a: Add, u: Unsubscribe, f: Fail, t: Transience, s: Start stream

- Nodes change state in each simulation step according to probabilities

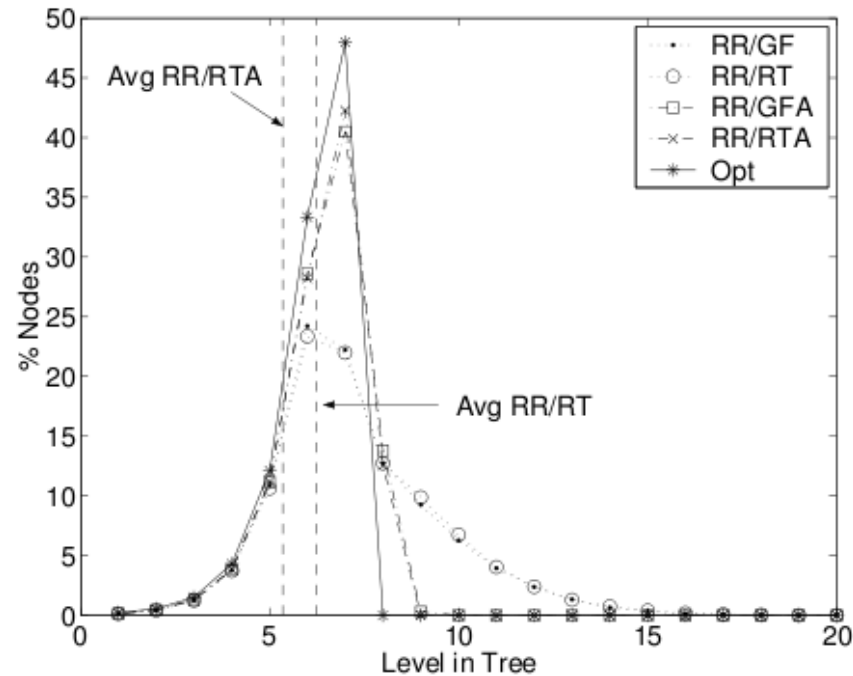
Join/Leave policies and cost of the tree



Notes: Opt is a hypothetical optimal policy

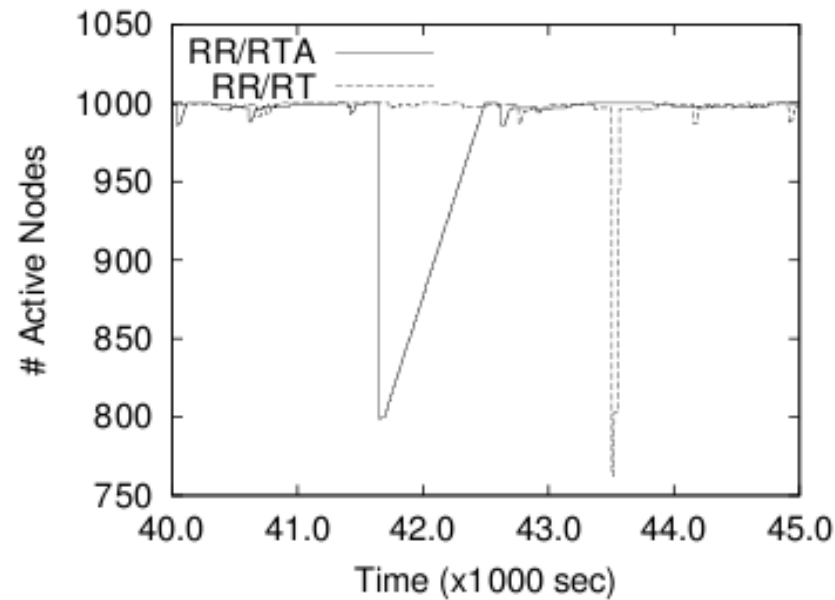
- Cost of the tree: $C_T = \sum_{i=1}^N h_i$ (h_i is the depth of node i)

Join/Leave policies and Tree shape



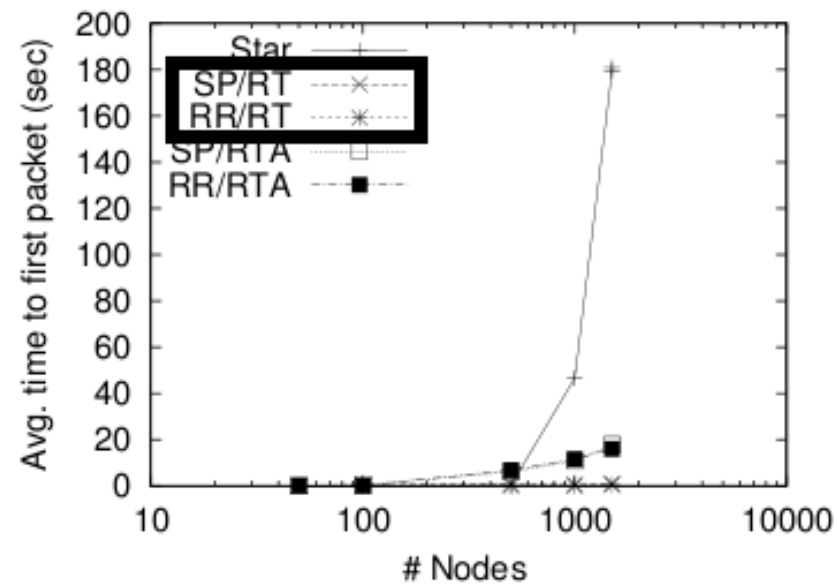
- The height is acceptable for all policies (≤ 20)

Drawback for RR and RTA



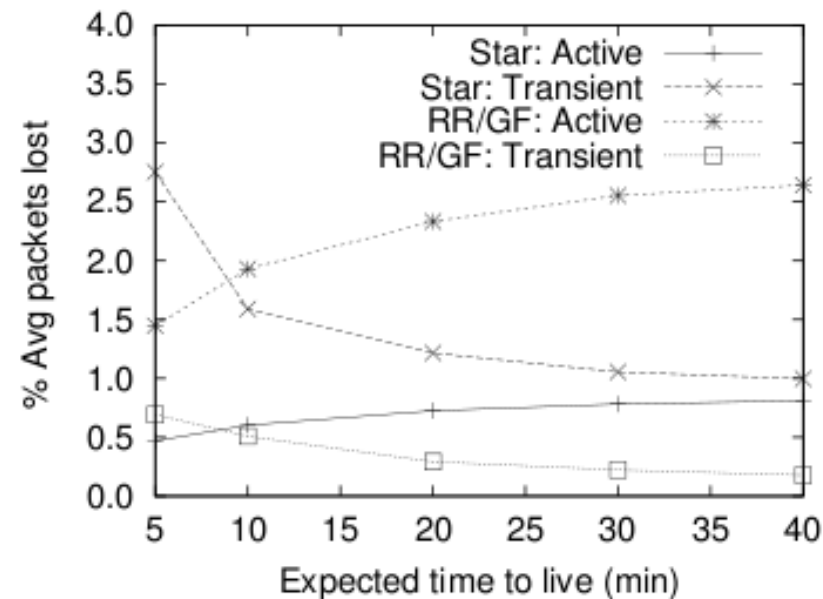
Notes: Transience is longer for RR and RTA policies (heavier maintenance burden)

Number of clients and TTFP



Notes: Star degrades as the number of requests increase

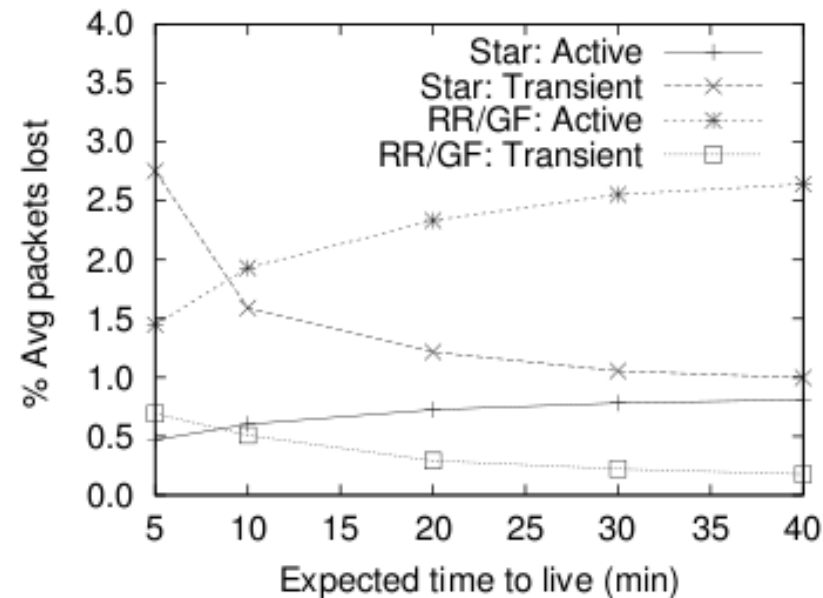
Packet losses



Notes: Unsubscribe rates and how they affect pkt losses

- RR/RT active: the more the peers stay alive, the more the tree tends to grow up and add delay (and drop probability) to a pkt during its traversal

Packet losses



Notes: Unsubscribe rates and how they affect pkt losses

- RR/RT active: the more the peers stay alive, the more the tree tends to grow up and add delay (and drop probability) to a pkt during its traversal

End System Multicast: Design goals

- Self-organization
 - Autonomous decentralized overlay construction
 - Adaptive to network/group dynamics
- Overlay efficiency
 - Minimal redundant transmission over physical link (network perspective)
 - Different applications may require different efficiency requirements
- Self-improving
 - The overlay must improve as new information is gathered

ESM workings

- Two-step procedure for the overlay construction
 - ① Build a mesh with the desired properties
 - ② Build spanning trees from each source to the interested group, using well-known algorithms
- Single, shared trees are single points of failure
- Building different overlays (one overlay for each tree spanning from each source) is an expensive alternative
- Trees built from a mesh reduce the overhead

ESM workings

- Two-step procedure for the overlay construction
 - ① Build a mesh with the desired properties
 - ② Build spanning trees from each source to the interested group, using well-known algorithms
- Single, shared trees are single points of failure
- Building different overlays (one overlay for each tree spanning from each source) is an expensive alternative
- Trees built from a mesh reduce the overhead

ESM workings

- Two-step procedure for the overlay construction
 - ① Build a mesh with the desired properties
 - ② Build spanning trees from each source to the interested group, using well-known algorithms
- Single, shared trees are single points of failure
- Building different overlays (one overlay for each tree spanning from each source) is an expensive alternative
- Trees built from a mesh reduce the overhead

ESM workings

- Two-step procedure for the overlay construction
 - ① Build a mesh with the desired properties
 - ② Build spanning trees from each source to the interested group, using well-known algorithms
- Single, shared trees are single points of failure
- Building different overlays (one overlay for each tree spanning from each source) is an expensive alternative
- Trees built from a mesh reduce the overhead

ESM workings

- Two-step procedure for the overlay construction
 - ① Build a mesh with the desired properties
 - ② Build spanning trees from each source to the interested group, using well-known algorithms
- Single, shared trees are single points of failure
- Building different overlays (one overlay for each tree spanning from each source) is an expensive alternative
- Trees built from a mesh reduce the overhead

Mesh optimization

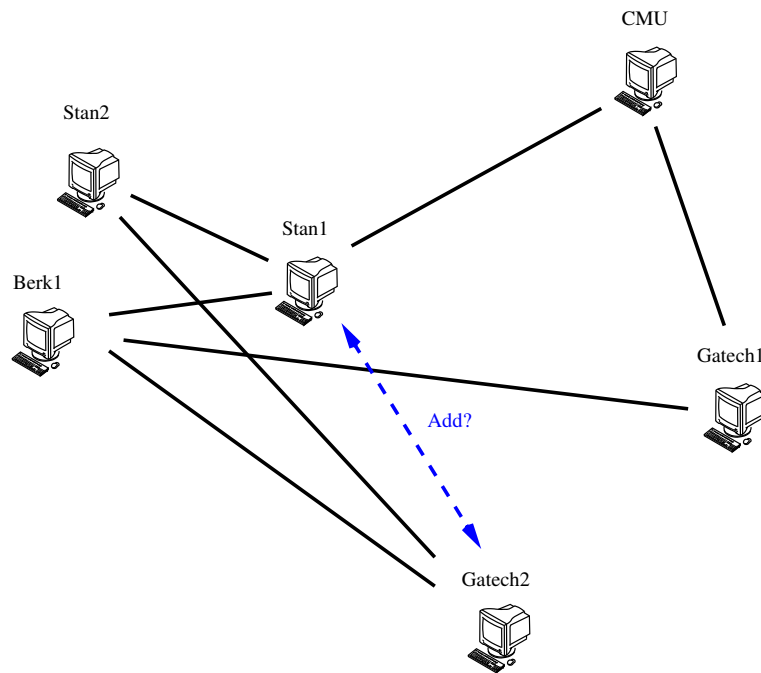
- Members periodically probe other members at random
- Definitions:
 - Utility gain (of adding a link): a function based on the number of members for which the delay gets better, and on how significant the improvement is for them
 - Cost of dropping a link between i and j : number of members for which i uses j as next hop
- New link added if

$$\textit{utilityGain} > \textit{addThreshold}$$

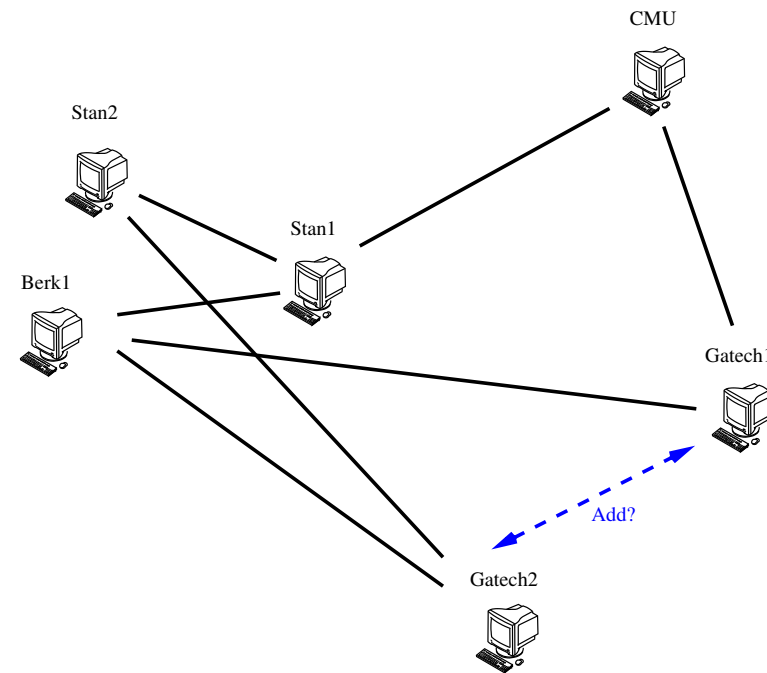
- Members periodically monitor existing links
- Existing link dropped if

$$\textit{consensusCost} < \textit{dropThreshold}$$

Mesh optimization example

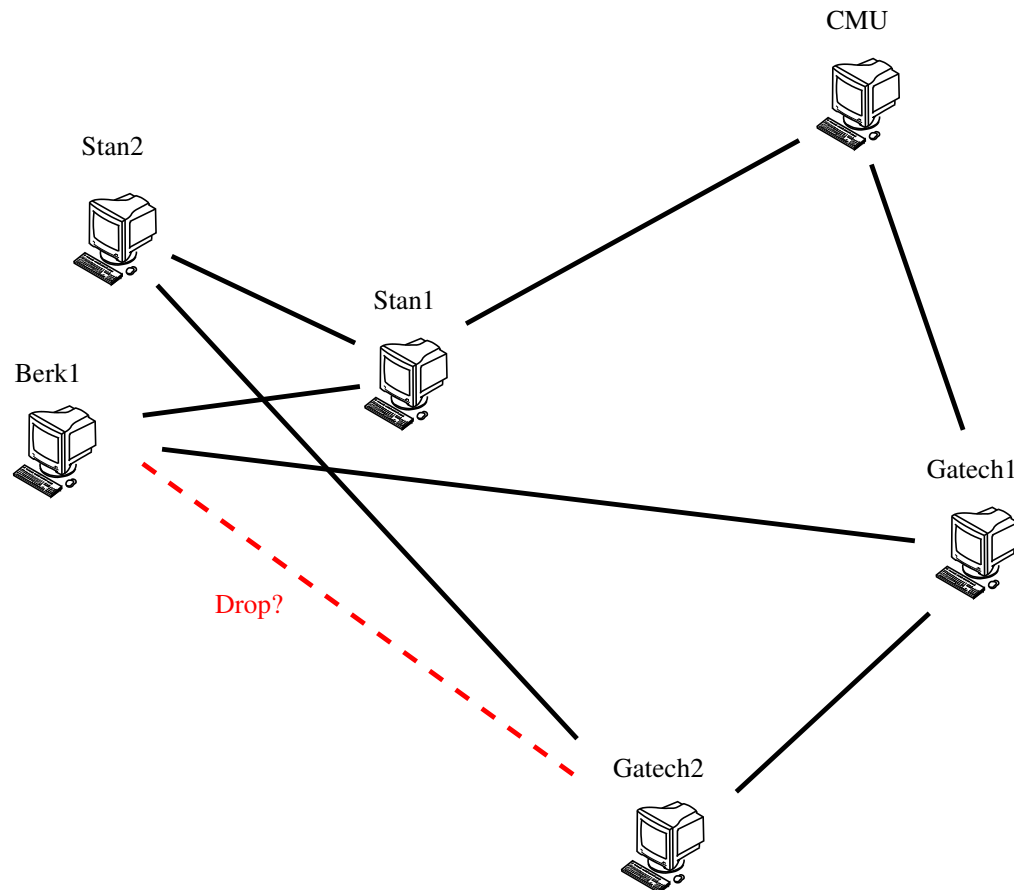


(a) Stan1-CMU delay improves, but marginally: Do not add link



(b) CMU-Gatech1 improves significantly: Add link

Mesh optimization example (2)



Notes: Drop because only Berk1 and Gatech2 benefit of this link, and an alternative exists

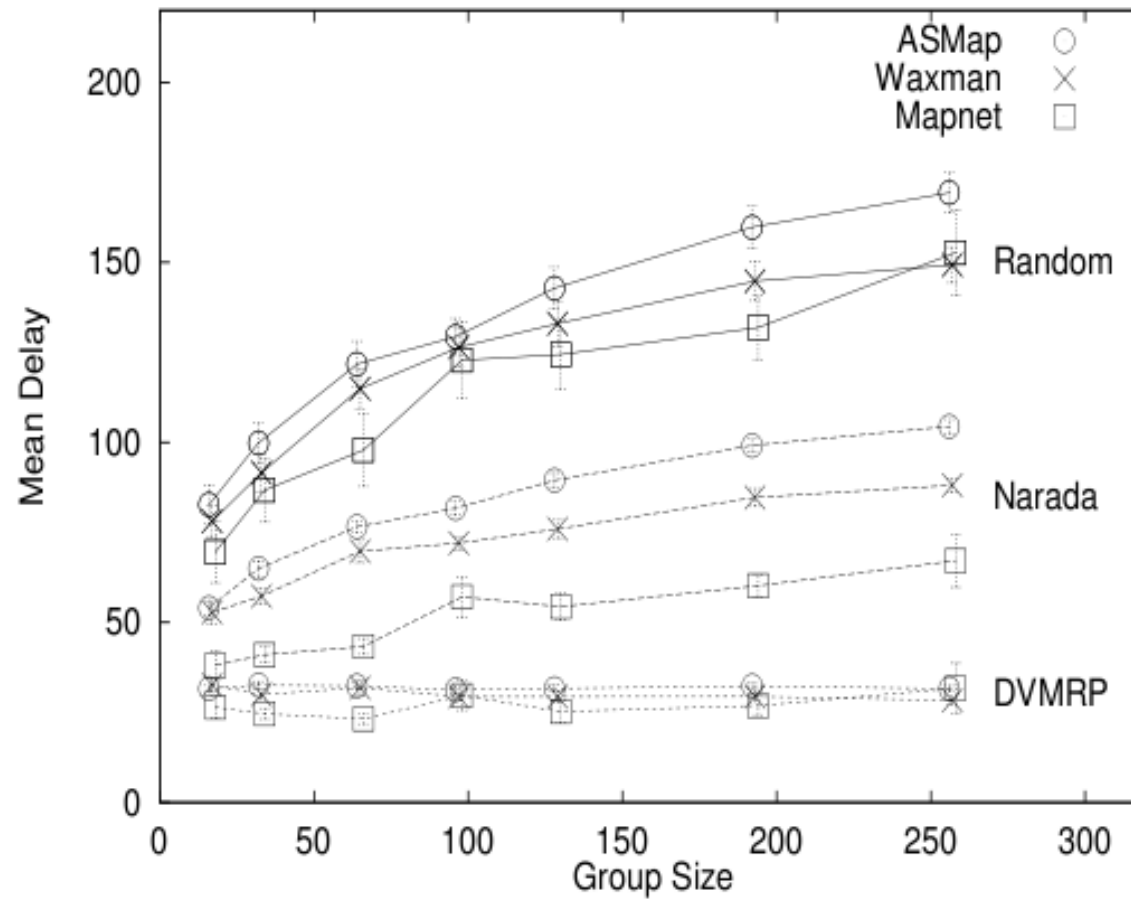
Evaluation

- Performance metrics:
 - Delay between members
 - Link stress: number of copies of the same packet traversing the same link
- Performance factors (affect the metrics):
 - Backbone topology model (Waxman, ASMap, Mapnet)
 - Topology size (64-1024 routers)
 - Group size (16-256)
 - FanOut range (neighbours maintained by a member)

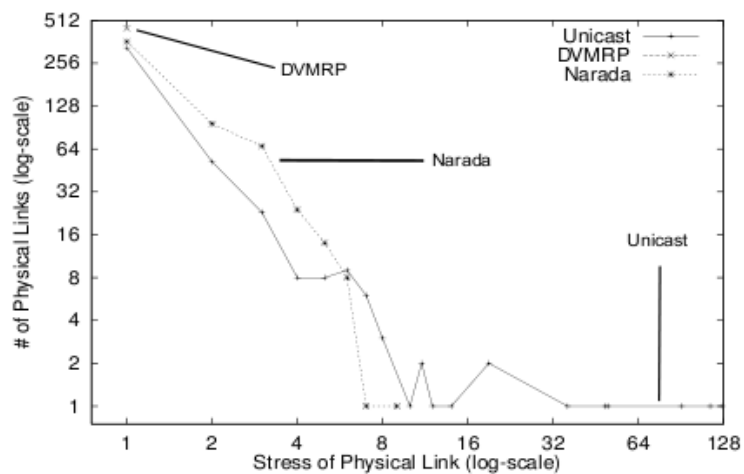
Evaluation

- Performance metrics:
 - Delay between members
 - Link stress: number of copies of the same packet traversing the same link
- Performance factors (affect the metrics):
 - Backbone topology model (Waxman, ASMap, Mapnet)
 - Topology size (64-1024 routers)
 - Group size (16-256)
 - FanOut range (neighbours maintained by a member)

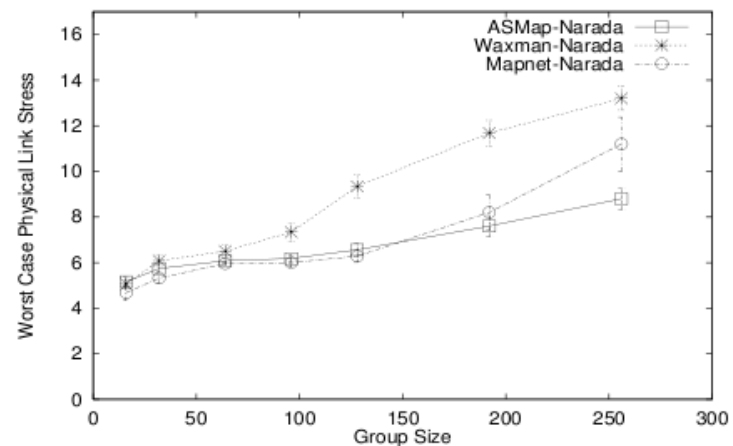
Delay



Link stress



(a) Link stress in the typical run



(b) Variation with topology model

CoolStreaming

- Real world deployed and highly successful system
- Goal: Live streaming
- Proprietary protocol
 - Results obtained by using reverse engineering
- Design Principles
 - A data-driven overlay (the overlay is determined by the availability of data)
 - Central authority (peerlist distribution, bootstrapping)
 - No global structure
 - Easy to implement; efficient; robust and resilient, because
 - 1 The absence of a global state to maintain keeps the implementation simple
 - 2 The directions of the swarm are determined by the availability of content: this means efficiency because there is only meaningful traffic
 - 3 Partners change supplier as node failures occur

CoolStreaming

- Real world deployed and highly successful system
- Goal: Live streaming
- Proprietary protocol
 - Results obtained by using reverse engineering
- Design Principles
 - A data-driven overlay (the overlay is determined by the availability of data)
 - Central authority (peerlist distribution, bootstrapping)
 - No global structure
 - Easy to implement; efficient; robust and resilient, because
 - 1 The absence of a global state to maintain keeps the implementation simple
 - 2 The directions of the swarm are determined by the availability of content: this means efficiency because there is only meaningful traffic
 - 3 Partners change supplier as node failures occur

CoolStreaming

- Real world deployed and highly successful system
- Goal: Live streaming
- Proprietary protocol
 - Results obtained by using reverse engineering
- Design Principles
 - A data-driven overlay (the overlay is determined by the availability of data)
 - Central authority (peerlist distribution, bootstrapping)
 - No global structure
 - Easy to implement; efficient; robust and resilient, because
 - 1 The absence of a global state to maintain keeps the implementation simple
 - 2 The directions of the swarm are determined by the availability of content: this means efficiency because there is only meaningful traffic
 - 3 Partners change supplier as node failures occur

CoolStreaming

- Real world deployed and highly successful system
- Goal: Live streaming
- Proprietary protocol
 - Results obtained by using reverse engineering
- Design Principles
 - A data-driven overlay (the overlay is determined by the availability of data)
 - Central authority (peerlist distribution, bootstrapping)
 - No global structure
 - Easy to implement; efficient; robust and resilient, because
 - 1 The absence of a global state to maintain keeps the implementation simple
 - 2 The directions of the swarm are determined by the availability of content: this means efficiency because there is only meaningful traffic
 - 3 Partners change supplier as node failures occur

CoolStreaming: Implementation

- Files are chopped by the server and disseminated in the swarm
- Upon arrival, a node obtains a peerlist of 40 nodes from the server
- Nodes contact the nodes in the peerlist for media content
- In steady state, every node has typically 4-8 neighbors, and it periodically shares its buffer map with neighbors
- Nodes exchange the unavailable content

CoolStreaming: Implementation

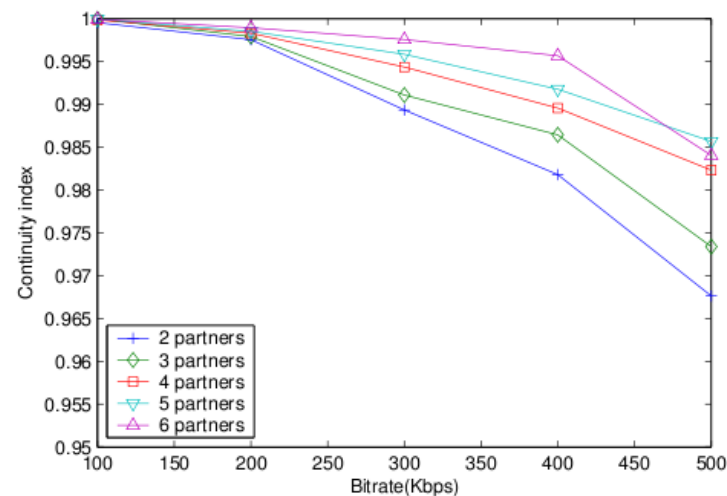
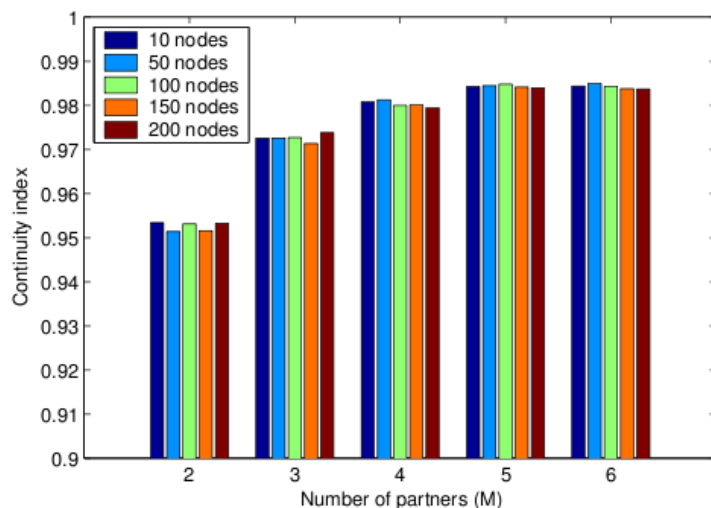
- Files are chopped by the server and disseminated in the swarm
- Upon arrival, a node obtains a peerlist of 40 nodes from the server
- Nodes contact the nodes in the peerlist for media content
- In steady state, every node has typically 4-8 neighbors, and it periodically shares its buffer map with neighbors
- Nodes exchange the unavailable content

CoolStreaming: Implementation

- Files are chopped by the server and disseminated in the swarm
- Upon arrival, a node obtains a peerlist of 40 nodes from the server
- Nodes contact the nodes in the peerlist for media content
- In steady state, every node has typically 4-8 neighbors, and it periodically shares its buffer map with neighbors
- Nodes exchange the unavailable content

CoolStreaming's Performances

- **Continuity index:** number of chunks arrived at destination before or exactly on the playback deadline
- The authors prove that the continuity index:
 - increases with the number of trading partners
 - decreases when the bitrate increases
 - gets worse when several short ON/OFF periods occur



A measurement of IPTV

- FIFA world cup session
 - 2 matches, the 1st using PPStream and SOPcast, the 2nd using PPLive and TVants
- tcpdump over two WinXP machines, 100 Mbps Ethernet access
- TCP session: first TCP pkt with payload to first TCP pkt after the pkt with end session flag (same procedure for UDP)

Brief reminder

NAME

tcpdump – dump traffic on a network

SYNOPSIS

```
tcpdump [ -AdDeflLnNOpqRStuUvxX ] [ -c count ]  
        [ -C file_size ] [ -F file ]  
        [ -i interface ] [ -m module ] [ -M secret ]  
        [ -r file ] [ -s snaplen ] [ -T type ] [ -w file ]  
        [ -W filecount ]  
        [ -E spi@ipaddr algo:secret ,... ]  
        [ -y datalinktype ] [ -Z user ]  
        [ expression ]
```

DESCRIPTION

Tcpdump prints out a description of the contents of packets on a network interface that match the boolean expression. It can also be run with the `-w` flag, which causes it to save the packet data to a file for later analysis, and/or with the `-r` flag, which causes it to read from a saved packet file rather than to read packets from a network interface. In all cases, only packets that match expression will be processed by tcpdump.

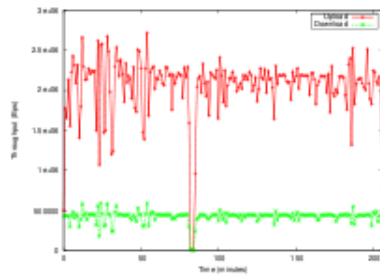
Traces summary

	Event	Trace size (MBytes)	Duration (seconds)	TCP UP (%)	TCP DOWN (%)	UDP UP (%)	UDP DOWN (%)
PPStream	game 1	4,121	12,375	79.50%	20.50%	none	none
SOPcast	game 1	5,475	12,198	3.89%	0.23%	79.98%	15.90%
PPlive	game 2	6,339	13,321	85.81%	14.09%	0.08%	0.02%
TVants	game 2	3,992	13,358	61.67%	14.71%	13.57%	10.05%

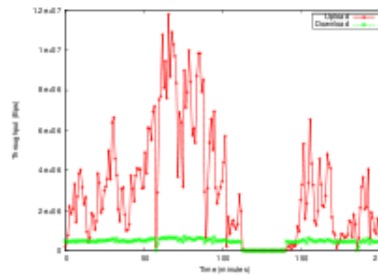
Notes: tcpdump aggregate output

- PPStream uses no UDP connections
- PPLive uses a few UDP connections

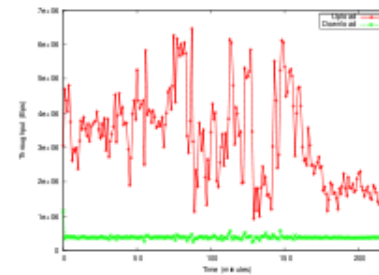
Bandwidth



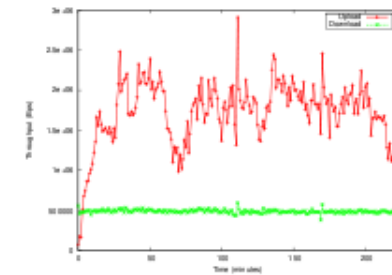
(a) PPStream



(b) SOPcast



(c) PPLive

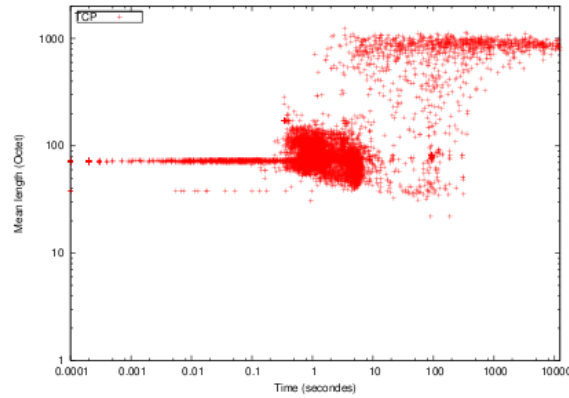


(d) TVants

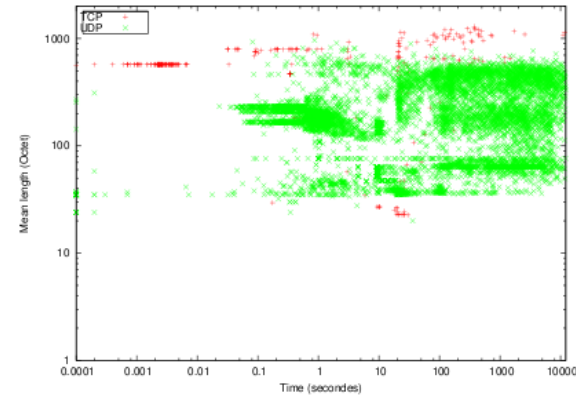
Notes: Total **download** (green) and **upload** (red) throughput

- Download steady, upload fluctuates
- Video source problem for SOPcast between min 130 and 140 (black screen)

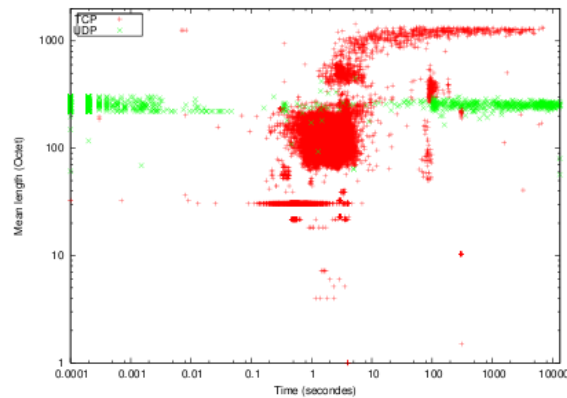
Traffic patterns



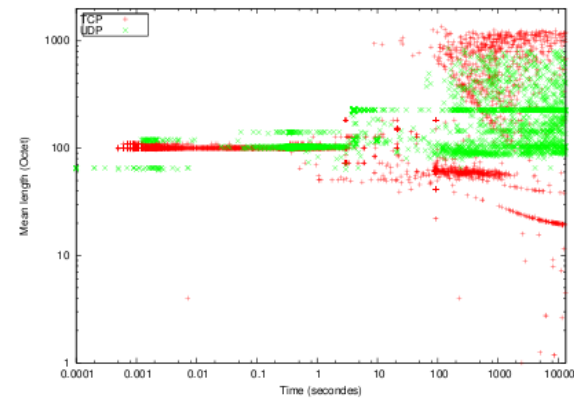
(a) PPStream



(b) SOPcast



(c) PPLive



(d) TVants

Notes: Avg pkt size vs. session duration (UDP green, TCP red)

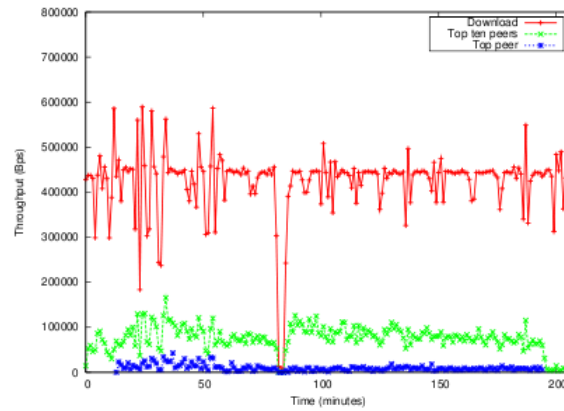
Traffic patterns: observations

- PPLive UDP sessions transport signalling traffic (short and constant sessions)
- PPLive and PPStream show clusters: middle is signalling, top-right is video
- TVants transport video both on TCP and UDP
- Heuristics to separate video from signalling: session with more than 10 *large* ($\geq 1000B$) pkts is a video session
- According to this heuristic, video traffic represents on avg the 80% of the overall traffic

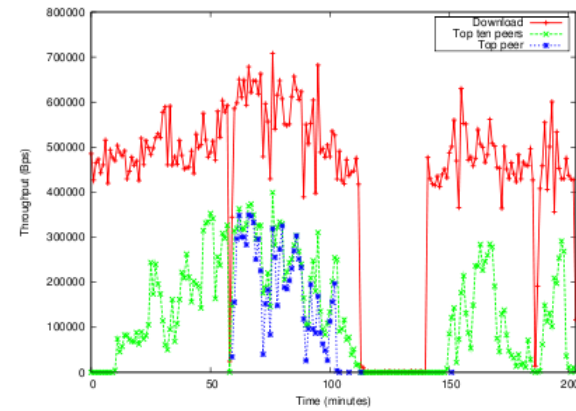
Traffic patterns: observations

- PPLive UDP sessions transport signalling traffic (short and constant sessions)
- PPLive and PPStream show clusters: middle is signalling, top-right is video
- TVants transport video both on TCP and UDP
- Heuristics to separate video from signalling: session with more than 10 *large* ($\geq 1000B$) pkts is a video session
- According to this heuristic, video traffic represents on avg the 80% of the overall traffic

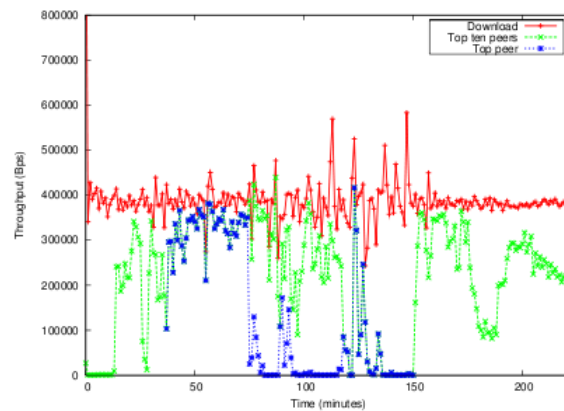
Download policies



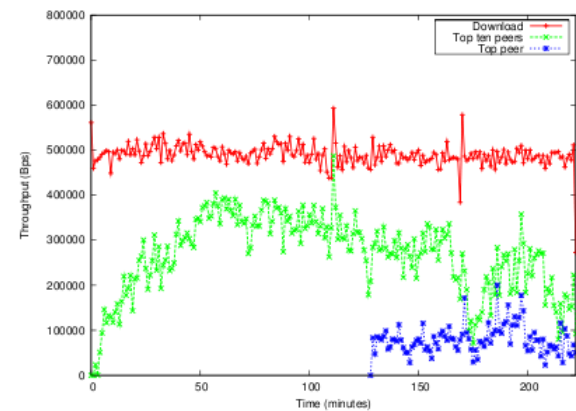
(a) PPStream



(b) SOPcast



(c) PPLive



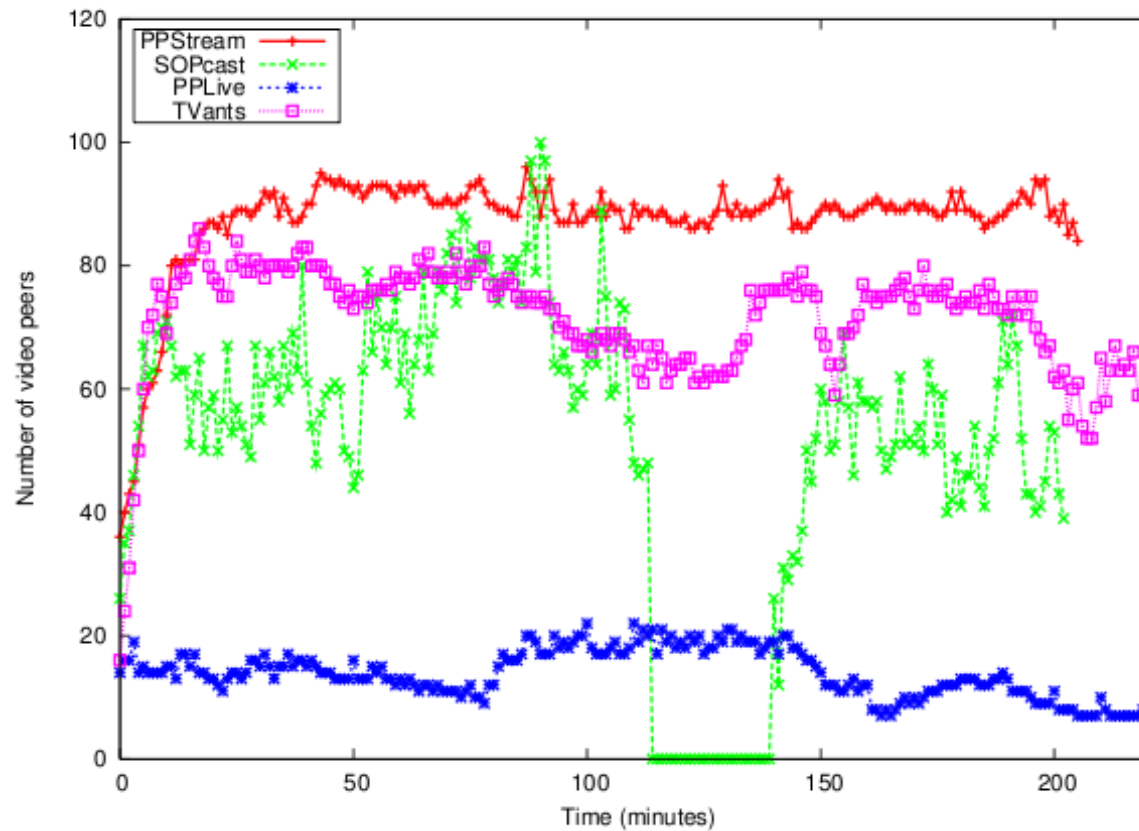
(d) TVants

Notes: Total traffic, top ten peers and top peer (bin duration: 60s)

Download policies: observations

- PPLive: periodical switch between peers (expect them to stay short in the network)
- PPStream:
 - top ten peers do not contribute much (compare with PPLive),
 - collects traffic from many peers at the same time
 - long sessions (expect peers to stay in the network for long time)
- SOPcast: top ten peers contribute for a half
- TVants:
 - top peer follows half transmission but contributes more than PPStream's top peer
 - half way between PPStream and SOPcast

Peers' behaviour and churn

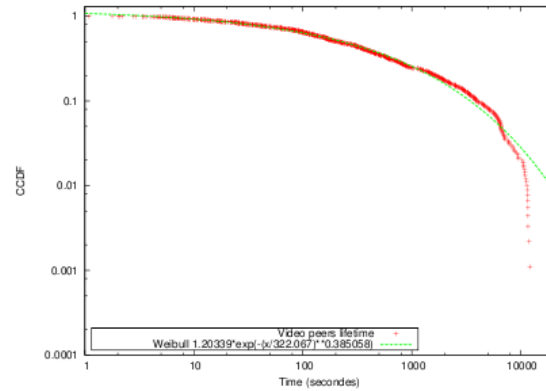


Notes: Churn of download peers

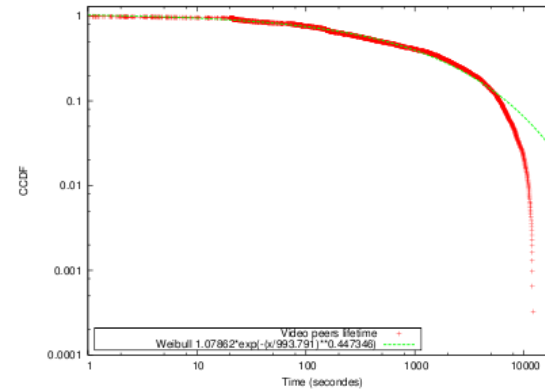
Peers' behaviour and churn: observations

- PPLive has a low but constant number of download peers (cf. PPStream)
- SOPcast's number of peers fluctuates
- PPStream has a download collapse @ min 90 but no peers leave the system (why?)
- What connection between churn and transport protocol (SOPcast uses mostly UDP, PPStream uses only TCP)?

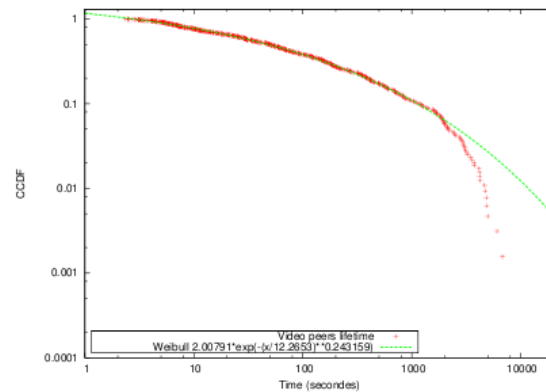
Peers' lifetime



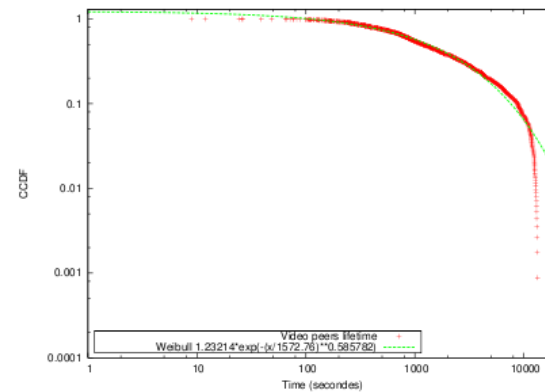
(a) PPStream. Average lifetime = 1222s



(b) SOPcast. Average lifetime = 1861s



(c) PPLive. Average lifetime = 393s



(d) TVants. Average lifetime = 2778s

Notes: Time peers stay in the system

References

- Video compression:
 - WAVE Report
(<http://www.wave-report.com/tutorials/VC.htm>)
 - Video/Imaging DesignLine (<http://www.videsignline.com>)
- Systems analyzed:
 - PeerCast: *Streaming live media over peers*, by H. Deshpande, M. Bawa, H. Garcia-Molina
 - ESM: *A case for end-system multicast*, by Y. Chu, S. G. Rao, S. Seshan, H. Zhang
 - CoolStreaming: *Coolstreaming/donet: A data-driven overlay network for efficient live media streaming*, by X. Zhang, J. Liu, B. Li, T. S. P. Yum
- Measurement paper:
 - *P2P IPTV measurement: a Comparison study*, by T. Silverston and O. Fourmaux