Dissemination–based Systems

- Applications
  - News services (RSS feeds)
  - Multiplayer network games
  - Real-time financial services

- Common facilities
  - Overlay network construction
  - Membership management
  - Data routing
  - Optimization
Application-aware Overlay Networks for Data Dissemination

O. Papaemmanouil, Y. Ahmad, U. Cetintemel, J. Jannotti

Brown University
Publish Subscribe Model

- Overlay tree construction
- Clients register
- Profiles *merging*
- Message *matching*
  - Message routing to interested clients
Application-specific Profile/Data Types

- Dissemination systems provide solutions for:
  - Message-Profile matching
  - Profile merging
  - Profile storage and indexing

- Existing approaches:
  - XML stream dissemination
    - e.g. ONYX[ref]
  - Relational data matching, storing and dissemination
    - e.g. SIENA [ref], GRYPHON [ref]
  - XPath profiles matching & storing
    - e.g. X Trie [ref], YFilter [ref]
Application-specific Cost Metrics

- Latency-related metrics
  - matching time, forwarding cost, latency

- Bandwidth-efficiency metrics
  - bandwidth consumption

- Fairness metrics
  - bandwidth utilization across nodes

- Reliability metrics
  - message loss rates
XPORT

- Distributed dissemination-based system
  - Supports extensibility in:
    - Profile and data management
      - match, merge, indexing
    - Performance goals
      - system cost, constraints
  - Provides metric-independent optimization
Profile Extensibility in XPORT

- Applications provide methods for handling their profile and data types

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>match ((message, profile))</td>
<td>Returns true if profile matches message</td>
</tr>
<tr>
<td>merge ((profile, profile))</td>
<td>Creates a more general profile</td>
</tr>
<tr>
<td>initializeIndex()</td>
<td>Initializes index</td>
</tr>
<tr>
<td>add ((profile))</td>
<td>Adds profile to the index structure</td>
</tr>
<tr>
<td>match ((index, profile))</td>
<td>Returns index entries matching profile</td>
</tr>
</tbody>
</table>
Cost Extensibility in XPORT

- Uses a two-level aggregation model
  - Level 1: Defines the *local* node cost
    - Aggregation of a metric of some neighbors
  - Level 2: Defines the *global* system cost
    - Aggregate costs of all nodes
- Provides grammar for metric specification
- Uses same model for metric constraints
Performance Metric Examples

- **Average path latency**
  - system cost = average (path latency)
  - path latency = sum (link latencies on path)

- **Bandwidth bottleneck**
  - system cost = min (path bandwidth)
  - path bandwidth = min (link bandwidth)

- **Variance of path latency**
  - system cost = variance (outgoing bandwidth consumption)
  - path bandwidth = sum (incoming data of all children)
Application-aware Optimization

- Applications define their optimization goals
  - Define node cost and system cost
  - Define constraint metrics and their thresholds
- Applications define their optimization rules
  - Composite new network reorganization steps
- XPORT customizes optimization to the application’s goals
  - Identifies beneficial network reorganizations with respect to these metrics
Optimization Protocol

- Divide network to sub-networks \((\text{optimization units})\)
  - One node responsible for each unit
  - Network units could overlap, but handled independently
- Consider transformations of each unit’s structure
  - Identify minimum cost \(local\) transformation
  - They should guarantee \(global\) cost improvement
- XPORT applies best \(local\) transformation among all units
  - Transformation with higher benefit on \(global\) system cost
  - Multiple units can be optimized in parallel
Network Transformations

- XPORT provides two primitive transformations
  - child promotion & child demotion
- Applications define composite transformations
  - e.g. subtree promotion & subtree demotion
Quantifying Cost Improvements

- Exploit semantics of aggregation functions to quantify a transformation’s cost benefit

- Use metric-independent cost model
  - Main idea: identify nodes affected by a transformation and maintain aggregated state to quantify their cost changes
  - Avoid any communication outside the optimization unit

- Based on function semantics automatically identify:
  - Stored state required
  - Metadata exchanged for optimization purposes
  - $O(1)$ for most aggregation functions
Example for sum of path latency

- Identify affected nodes
  - Link latency changes are reflected on the descendants’ latency
- Node state required
  - Size of subtree
- Total cost change
  - $\text{cost change} \times \text{subtree size}$
- No communication required with affected nodes

System cost =\((2-7) \times 2 + (7-2) \times 1 = 5\)
Optimization Approaches

- **Bottleneck-based**
  - Optimize global cost
    - max CPU load in network
  - Optimize critical units
    - Units with max CPU load
  - Guarantees improvement
  - Reactive optimization
    - Work on units with most loaded node

- **Opportunistic-based**
  - Optimize cost of unit
    - max CPU load in unit
  - Optimize all units
    - Independent from CPU load
  - No improvement guarantees
  - Proactive optimization
    - Could prevent nodes from increasing their CPU load
RSS Feed dissemination

- XPORT distributes RSS feeds
- Clients connect to a proxy and send their requests
- Root periodically polls RSS sources
- Benefits:
  - RSS sources experience less load
  - RSS clients receive more timely updates
Deployment on Planet Lab

XPORT converges to the optimal topology after 8 transformations
Future Work

- Stateful subscriptions
- Message customization
- Collection & dissemination integration
- High-level profile languages
- Support overlay meshes
Stateful Subscriptions

- “Drop similar messages if they arrive within less than 10 secs”
- *match* method should maintain state
  - e.g. messages of last 10 secs
  - Global state per application
    - Operations have similar semantics for each client
  - State per profile
    - Clients specify independent stateful operations
Incoming Message Customization

- Transform message structure and values
  - Map message values to different domains
  - e.g. XPath profiles
- `match` method will support data transformation
  - Returns the output message
Collection & Dissemination Integration

- Create a collection network to pull data from sources
  - Distribute overhead to multiple nodes
- Define new primitive transformations
  - Assign part of my sources to my parent
- Combined optimization
  - Short circuit
High-level Profile Languages

- Complex profile types are hard to specify
  - Give me those news that do not look like the ones I received during the last two days

- XPORT will provide native profile language
  - Built-in stream-oriented operators
  - Users can express complex profiles
  - XPORT automatically derives the API methods based on the operators’ semantics
Support Overlay Meshes

- Create and optimize an overlay mesh
  - Improves throughput and reliability
- Extend optimization framework
  - 3-level aggregation
    - Global cost is the aggregation of each tree cost
  - Improve cost across multiple trees
Conclusions

- We explored extensibility in overlay routing trees
  - Profile-based data dissemination systems
- We designed and implemented XPORT
  - Application-aware dissemination-based infrastructure
  - Highly extensible and application customizable