Single Source Multiple Destinations Data Transmission in the Internet

Piotr Srebrny
Part I: Multicasting in the Internet
  - Basis & critique

Part II: CacheCast
  - Internet redundancy
  - Packet caching systems
  - CacheCast design
  - CacheCast evaluation
    - Efficiency
    - Computational complexity
    - Environmental impact
Multicasting in the Internet
In the beginning there was the Internet...
...and the Internet was multicast-less
Growing trees in the Internet
Why not to write all the destination into one packet?
Hi,
I would like to invite you to the presentation on the IP Multicast issues.

DMNS Group
...and the resulting types of trees differ very...
We want the multicast that achieves enormous scalability!!

Scalability was like the Holy Grail for multicast community.
The first multicast service model: Any Source Multicast (ASM)
The first multicast service model: Any Source Multicast (ASM)
Limited number of channels that were not allocated...
... it lacked congestion control mechanism...
...it was open...
... and it needed full deployment to work.
So, IP Multicast evolved...

... and after 10 years of non-random mutations
Single Source Multicast
But still aiming at the enormous scalability.
What is IP Multicast?

- MBGP
- BGP4+
- IGMP Snooping
- CGMP
- RGMP
- AMT
- PGM
- MOSPF
- PIM-SM
- PIM-SSM
- PIM-DM
- DVMRP
- IGPMv2
- IGPMv3
- MLDv2
- MLDv3
- MADCAP
- MASC
- MAAS
- MSDP
‘Growth’ in Multicast Deployement

The Percentage of the Internet Supporting Multicast

http://multicasttech.com/status/ (obsolete)
Inherited problems of the multicast architecture
Did IP Multicast fail?
CacheCast

Part II
Part II: Outline

- Internet redundancy
- Packet caching systems
- CacheCast design
- CacheCast evaluation
  - Efficiency
  - Computational complexity
  - Environmental impact
- Related system
- Summary
Internet of Content

- Internet is a content distribution network

---

**Distribution of protocol classes 2008/2009**

<table>
<thead>
<tr>
<th>Region</th>
<th>P2P</th>
<th>Standard</th>
<th>Web</th>
<th>Streaming</th>
<th>VoIP</th>
<th>Gaming</th>
<th>Unknown</th>
<th>IM</th>
<th>Tunnel</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Africa</td>
<td>66%</td>
<td>21%</td>
<td>6%</td>
<td>5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S. America</td>
<td>65%</td>
<td>18%</td>
<td>8%</td>
<td>7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. Europe</td>
<td>70%</td>
<td>16%</td>
<td>7%</td>
<td>6%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>53%</td>
<td>26%</td>
<td>7%</td>
<td>8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S. Europe</td>
<td>55%</td>
<td>25%</td>
<td>10%</td>
<td>9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Middle East</td>
<td>45%</td>
<td>34%</td>
<td>5%</td>
<td>10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N. Africa</td>
<td>43%</td>
<td>33%</td>
<td>9%</td>
<td>14%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sw. Europe</td>
<td>54%</td>
<td>23%</td>
<td>10%</td>
<td>9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Content Transport

- Single source multiple destination transport mechanism becomes fundamental!
- At present, Internet does not provide efficient multi-point transport mechanism
Former Efforts

- “Host groups: A multicast extension for datagram internetworks”, D. Cheriton and S. Deering, 1985 – destination address denotes a group of host
- “A case for end system multicast”, Y. hua Chu et al., 2000 – application layer multicast
Source of Redundancy

- Server transmitting the same data to multiple destinations is wasting the Internet resources
  - The same data traverses the same path multiple times
Consider two packets A and B that carry the same content and travel the same few hops.
Consider two packets A and B that carry the same content and travel the same few hops.
In practice:

- How to determine whether a packet payload is in the next hop cache?
- How to compare packet payloads?
- What size should be the cache?
CacheCast Basics
Network elements:

- **Link**
  - Medium transporting packets

- **Router**
  - Switches data packets between links
**Link**

- Link
  - Logical point to point connection
  - Highly robust & very deterministic
  - Throughput limitation per bit [bps]

➢ It is beneficial to avoid redundant payload transmissions over a link
Router

- Switching node
- Performs three elementary tasks per packet
  - TTL update
  - Checksum recalculation
  - Destination IP address lookup
- Throughput limitation per packet [pps]

- Forwarding packets with redundant payload does not impact router performance
Link Cache

- Caching is done on per link basis
- Cache Management Unit (CMU) removes payloads that are stored on the link exit
- Cache Store Unit (CSU) restores payloads from a local cache
Link Cache – Requirements

- Link cache processing must be simple
  - ~72ns to process the minimum size packet on a 10Gbps link
  - Modern memory r/w cycle ~6-20ns
- Link cache size must be minimised
  - At present, a link queue is scaled to 250ms of the link traffic, for a 10Gbps link it is already 315MB
  - Difficult to build!

A source of redundant data must support link caches!
1. Server can transmit packets carrying the same data within a minimum time interval

2. Server can mark its redundant traffic

3. Server can provide additional information that simplifies link cache processing
CacheCast Packet Header

- CacheCast packet carries an extension header describing packet payload
  - Payload ID
  - Payload size
  - Index
- Only packets with the header are cached

Payload size

| Payload | • | • | • | L3 | Payload size | Payload id | INDEX | L2 |
Packet train

- Only the first packet carries the payload
- The remaining packets truncated to the header
Packet train duration time

- It is sufficient to hold payload in the CSU for the packet train duration time.
- What is the maximum packet train duration time?
## Link Cache Size (cont.)

### Back-of-the-envelope calculations

<table>
<thead>
<tr>
<th>Source uplink speed</th>
<th>Cache hold time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2ms</td>
</tr>
<tr>
<td>512Kbps</td>
<td>2</td>
</tr>
<tr>
<td>1Mbps</td>
<td>4</td>
</tr>
<tr>
<td>10Mbps</td>
<td>32</td>
</tr>
<tr>
<td>100Mbps</td>
<td>313</td>
</tr>
</tbody>
</table>

~10ms caches are sufficient
CacheCast System

Two components of the CacheCast system

- Server support
- Distributed infrastructure of small link caches
CMU & CSU

- Cache miss

<table>
<thead>
<tr>
<th>Index</th>
<th>Payload ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P2</td>
</tr>
<tr>
<td>1</td>
<td>P3</td>
</tr>
<tr>
<td>2</td>
<td>P4</td>
</tr>
</tbody>
</table>

CMU table

<table>
<thead>
<tr>
<th>Index</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P2</td>
</tr>
<tr>
<td>1</td>
<td>P3</td>
</tr>
<tr>
<td>2</td>
<td>P4</td>
</tr>
</tbody>
</table>

Cache store
• Cache hit

CMU & CSU (cont.)

**CMU table**

<table>
<thead>
<tr>
<th>Index</th>
<th>Payload ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P1</td>
</tr>
<tr>
<td>1</td>
<td>P2</td>
</tr>
<tr>
<td>2</td>
<td>P3</td>
</tr>
</tbody>
</table>

**Cache store**

<table>
<thead>
<tr>
<th>Index</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P1</td>
</tr>
<tr>
<td>1</td>
<td>P2</td>
</tr>
<tr>
<td>2</td>
<td>P3</td>
</tr>
</tbody>
</table>
Errors on Link

• Cache miss

What can go wrong?
Errors on Link (cont.)

- Cache hit

**CMU table**

<table>
<thead>
<tr>
<th>Index</th>
<th>Payload ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P1</td>
</tr>
<tr>
<td>1</td>
<td>P3</td>
</tr>
<tr>
<td>2</td>
<td>P4</td>
</tr>
</tbody>
</table>

**Cache store**

<table>
<thead>
<tr>
<th>Index</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P2</td>
</tr>
<tr>
<td>1</td>
<td>P3</td>
</tr>
<tr>
<td>2</td>
<td>P4</td>
</tr>
</tbody>
</table>

How to protect against this error?
Server Support

- **Tasks:**
  - Batch transmissions of the same data to multiple destinations
  - Build the CacheCast headers
  - Transmit packets in the form of a *packet train*

- One system call to transmit data to all destinations
  - `msend()`
- **msend()** system call
  - Implemented in Linux
  - Simple API

```c
int msend(fds_set *fds_write,
        fds_set *fds_written,
        char *buf, int len)
```

- `fds_write` – a set of file descriptors representing connections to data clients
- `fds_written` – a set of file descriptors representing connections to clients that the data was transmitted to
Server Support (cont.)

- OS network stack
  - Connection endpoints represented as sockets
  - Transport layer (e.g. TCP, UDP, or DCCP)
  - Network layer (e.g. IP)
  - Link layer (e.g. Ethernet)
  - Network card driver
Server Support (cont.)

- `msend()` execution

![Diagram showing msend execution process with steps labeled (1) connect(), (2) msend(), (3) transmit, and (4) transmit with corresponding time line.]

CacheCast Evaluation
CacheCast Feasibility

Two aspects of the CacheCast system

I. Efficiency
   ▪ How much redundancy CacheCast removes?

II. Computational complexity
   ▪ Can CacheCast be implemented efficiently with the present technology?
CacheCast and ‘Perfect multicast’

- ‘Perfect multicast’ – delivers data to multiple destinations without any overhead

CacheCast overheads

1. Unique packet header per destination
2. Finite link cache size resulting in payload retransmissions
3. Partial deployment
Efficiency – Metric

\[ \delta_m = 1 - \frac{L_m}{L_u} \]

- \( L_m \) - the total amount of multicast links
- \( L_u \) - the total amount of unicast links

Metric expresses the reduction in traffic volume

- Example:

\[ L_u = 9, \; L_m = 5 \]

\[ \delta_m = 1 - \frac{5}{9} = \frac{4}{9} \approx 44\% \]
Efficiency – Unique Packet Headers

CacheCast unicast header part \( (h) \) and multicast payload part \( (p) \)

\[ \delta_m = 1 - \frac{L_m}{L_u} \implies \delta_{CC} = 1 - \frac{s_h L_u + s_p L_m}{(s_h + s_p) L_u} \]

Thus:

\[ \delta_{CC} = \frac{1}{1 + r} \delta_m, \quad r = \frac{s_h}{s_p} \]

E.g.:

using packets where \( s_p = 1436B \) and \( s_h = 64B \), CacheCast achieves 96% of the ‘perfect multicast’ efficiency
Efficiency – Single Link Cache

- Single link cache efficiency is related to the amount of redundancy that is removed

$$\eta = 1 - \frac{V_c}{V}$$

\[V_c - \text{traffic volume with CacheCast}\] 
\[V - \text{traffic volume without CacheCast}\]

- Traffic volumes:
  - \(V = n(s_p + s_h)\)
  - \(V_c = s_p + n s_h\)
Efficiency – Single Link Cache

- Link cache efficiency:

\[ \eta = 1 - \frac{s_p + n s_h}{n s_p + n s_h} \]

- Thus:

\[ \eta = \left( 1 - \frac{1}{n} \right) C \quad \left| C = \frac{1}{1 + r}, r = \frac{s_h}{s_p} \right. \]
Efficiency – Single Link Cache (cont.)

\[ \eta = \left( 1 - \frac{1}{n} \right) C \]
Finite Cache Size

- The more destination the higher efficiency
- E.g.
  - 512Kbps – 8 headers in 10ms, e.g. 12 destinations
  - Slow sources transmitting to many destinations cannot achieve the maximum efficiency
Efficiency of the CacheCast System

System efficiency $\delta_m$ for 10ms large caches

<table>
<thead>
<tr>
<th>Source uplink speed</th>
<th>10ms packet train size</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 Kbps</td>
<td>8</td>
</tr>
<tr>
<td>1 Mbps</td>
<td>16</td>
</tr>
<tr>
<td>10 Mbps</td>
<td>157</td>
</tr>
<tr>
<td>100 Mbps</td>
<td>1561</td>
</tr>
</tbody>
</table>
Source of Inefficiency

- Step I
Source of Inefficiency

- Step II
Source of Inefficiency

- Step III
Source of Inefficiency

- Step IV

- How could we improve?
CMU and CSU deployed partially
Efficiency – Incremental Deployment

![Graph showing the percentage of efficiency against the number of hops covered by caches. The x-axis represents the number of hops, ranging from 1 to 19. The y-axis represents the percentage of efficiency, ranging from 10% to 100%. The bars are color-coded: 10, 100, and 1000, indicating different datasets or configurations. The graph illustrates how the percentage of efficiency increases with the number of hops covered by caches.]
Considering unique packet headers
- CacheCast can achieve up to 96% of the ‘Perfect multicast’ efficiency

Considering finite cache size
- 10ms link caches can remove most of the redundancy generated by fast sources

Considering partial deployment
- CacheCast deployed over the first five hops from a server achieves already half of the maximum efficiency
Computational Complexity
Computational Complexity

- Computational complexity may render CacheCast inefficient

- Implementations
  - Link cache elements – implemented with Click Modular Router Software as processing elements
  - Server support – a Linux system call and an auxiliary shell command tool
CacheCast Router

- CacheCast can be deployed as a software update
  - Click Modular Router Software

- CacheCast router modell
Router configuration:
- CSU – first element
- CMU – last element
- Packet drop occurs at the output link queue, however before CMU processing
Workload

- Packet trains

Payload sizes: 500B, 1500B, 9000B, 16000B

Group size: 10, 100, 1000
Due to CSU and CMU elements CacheCast router cannot forward packet trains at line rate.
When compared with a standard router, CacheCast router can forward more data.

<table>
<thead>
<tr>
<th>Group size</th>
<th>Payload size 500B</th>
<th>Payload size 1500B</th>
<th>Payload size 9000B</th>
<th>Payload size 16000B</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4.49</td>
<td>7.35</td>
<td>9.42</td>
<td>9.67</td>
</tr>
<tr>
<td>100</td>
<td>5.82</td>
<td>13.36</td>
<td>34.37</td>
<td>40.11</td>
</tr>
<tr>
<td>1000</td>
<td>5.80</td>
<td>13.52</td>
<td>35.38</td>
<td>40.10</td>
</tr>
</tbody>
</table>
Efficiency of the server support is related to the efficiency of the `msend()` system call.

Basic metric:
- Time to transmit a single packet
- Compare the `msend()` system call with the standard `send()` system call
Server transmitting to 100 destinations using
- Loop of `send()` sys. calls
- A single `msend()` sys. call

- `msend()` system call outperforms the standard `send()` system call when transmitting to multiple destinations
Audio Streaming

- Paraslash audio streaming software

http://paraslash.systemlinux.org/

dccp_send.c

```c
while (written < len) {
    size_t num = len - written;
    if (num > DCCP_MAX_BYTES_PER_WRITE) {
        num = DCCP_MAX_BYTES_PER_WRITE;
    }
    msend(&dss->client_fds, &fdw, buf, num);
    written += num;
}
/* We drop chunks that we don't manage to send */
```
Audio Streaming (cont.)

- Setup

- Paraslash clients located at the machines A and B gradually request an audio stream from the server S
Audio Streaming (cont.)

- Original *paraslash* server can only handle 74 clients
- CacheCast *paraslash* server can handle 1020 clients and more depending on the chunk size
- Server load is reduced when using large chunks

<table>
<thead>
<tr>
<th>Server type</th>
<th>Chunk size</th>
<th>Users</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1024</td>
<td>74</td>
<td>2.71 ± 1.1</td>
</tr>
<tr>
<td>CacheCast</td>
<td>1024</td>
<td>1020</td>
<td>44.21 ± 1.6</td>
</tr>
<tr>
<td>CacheCast</td>
<td>2066</td>
<td>2066</td>
<td>59.63 ± 1.7</td>
</tr>
<tr>
<td>CacheCast</td>
<td>4184</td>
<td>4097</td>
<td>58.30 ± 1.0</td>
</tr>
<tr>
<td>CacheCast</td>
<td>8364</td>
<td>8001</td>
<td>87.17 ± 1.8</td>
</tr>
<tr>
<td>CacheCast</td>
<td>15674</td>
<td>12454</td>
<td>91.35 ± 3.049</td>
</tr>
</tbody>
</table>
Environmental impact
Internet congestion avoidance relies on communicating end-points that adjust transmission rate to the network conditions.

CacheCast transparently removes redundancy increasing network capacity.

It is not given how congestion control algorithms behave in the CacheCast presence.
- CacheCast implemented in _ns-2_
- Simulation setup:
  - Bottleneck link topology
  - 100 TCP flows and 100 TFRC flows
  - Link cache operating on a bottleneck link
Environmental Impact (cont.)

- TCP flows consume the spare capacity
- TFRC flows increase end-to-end throughput

CacheCast preserves the Internet ‘fairness’
“Packet caches on routers: the implications of universal redundant traffic elimination” Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan, and Scott Shenker, SIGCOMM’08

- Fine grain redundancy detection
  - 10-50% removed redundancy
- New redundancy aware routing protocol
  - Further 10-25% removed redundancy
- Large caches
  - Caching 10s of traffic traversing a link
IP Multicast
- Based on the host group model to achieve great scalability
- Breaks end-to-end model of the Internet communication
- Operates only in ‘walled gardens’

CacheCast
- Only removes redundant payload transmission and preserves end-to-end connections
- Can achieve near multicast bandwidth savings
- Is incrementally deployable
- Preserves fairness in Internet
- Requires server support

A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, “Packet caches on routers: the implications of universal redundant traffic elimination,” *SIGCOMM’08*

J. Santos and D. Wetherall, “Increasing effective link bandwidth by suppressing replicated data,” *USENIX’98*