Efficient Operating System Support for Group Unicast

Martin Karsten, Jialin Song, Michael Kwok, Tim Brecht
Problem

Some apps send same data to multiple receivers

Sending to group of red hosts/users
Example Applications

– Distributed Virtual Environments
  • Multiplayer on-line games
  • Computer Supported Cooperative Work (CSCW)
– Audio/Video conferencing
– Chat room servers
– Streaming media servers
– Multicast overlay networks
Example Applications

– Distributed Virtual Environments
  • Multiplayer on-line games
  • Computer Supported Cooperative Work (CSCW)
– Audio/Video conferencing
– Chat room servers
– Streaming media servers
– Multicast overlay networks

Many/most of these use UDP

How to efficiently send to a group using UDP?
Example Applications

– Distributed Virtual Environments
  • Multiplayer on-line games
  • Computer Supported Cooperative Work (CSCW)
– Audio/Video conferencing
– Chat room servers
– Streaming media servers
– Multicast overlay networks

Many/most of these use UDP

How to efficiently send to a group using UDP?

Other Transport Protocols ➔ Future Work
Possible Approaches / Related Work

- **IP Multicast**
  - difficulties in wide spread deployment (not feasible)

- **Multicast Overlay networks**
  - implemention requires group communication

- **Common Approach: User-level unicast (user-groupcast)**
Possible Approaches / Related Work

- **IP Multicast**
  - difficulties in wide spread deployment (not feasible)

- **Multicast Overlay networks**
  - implementation requires group communication

- **Common Approach: User-level unicast (user-groupcast)**

```c
for (i=0; i<GRPSIZE; i++) {
    fds[i] = socket(PF_INET, SOCK_DGRAM, 0);
}
for (i=0; i<GRPSIZE; i++) {
    bytes += send(fds[i], buf, bytes);
}
```
Kernel-Level Group Unicast (kernel-groupcast)

```
grp = socket(PF_INET, SOCK_DGRAM, 0);
setsockopt(grp, SOL_SOCKET,
    SO_SETGRP, addrs,
    GRPSIZE * sizeof(struct sockaddr_in));
bytes = send(grp, buf, bytes);
```
Implementation Overview

- **User level**
  - payload
  - socket
  - packet buffer
  - socket

- **Kernel level**
  - address list

- **UDP layer**
  - repeated packet formation

- **IP Layer**

- **Ethernet layer**
  - output queue
  - DMA
  - NIC
Experimental Environment

- **Server**: 400 MHz Pentium II, 2 x e1000 Gbps enet
  - FreeBSD 5.2.1, Fedora Core 2 with 2.6.8 kernel
- **Switch**: HP Procurve Gbps switch: 24 ports
- **Clients**: 550 MHz Pentium III

Deliberately set up so that sender is bottleneck
FreeBSD Micro-benchmark: 100 bytes

![Graph showing send time for different group sizes]

- **user/100**
- **kernel/100**
FreeBSD Micro-benchmark : 1000 bytes

Send Time (ms)

Group Size

user/1000

kernel/1000
FreeBSD Micro-benchmark: with grp change
Software Slicing: User-groupcast

- Payload
- User level
- Socket layer
- Packet buffer
- Kernel level
- Socket address list
- UDP layer
- UDP processing
- Repeated packet formation
- IP Layer
- Ethernet layer
- Output queue
- DMA
- NIC

- Start of memcpy
- End of memcpy
- End UDP / Start IP
- Start of driver
- Complete send
Software Slicing: User-groupcast

- Start of memcpy
- End of memcpy
- End UDP / Start IP
- Start of driver
- Complete send
Software Slicing: User-groupcast

start of memcpy

end of memcpy

end UDP / start IP

start of driver

complete send
Software Slicing: User-groupcast

1. Start of memcpy
2. End of memcpy
3. End UDP / Start IP
4. Start of driver
5. Complete send
Software Slicing: User-groupcast

- start of memcpy
- end of memcpy
- end UDP / start IP
- start of driver
- complete send
User-Level Send Cost Breakdown: FreeBSD

![Graph showing normalized cost versus packet size for different stages of sending packets in FreeBSD.](image)
User-Level Send Cost Breakdown: FreeBSD

kernel or user-groupcast ~1 interrupt per system call

ether_output() 10 x faster: kernel-groupcast
Is this Important/Relevant to Applications?

- decrease latencies
- increase number of users / recipients
## Increase Group Size: (100 bytes, 33.3 ms)

<table>
<thead>
<tr>
<th>$N$</th>
<th>User send</th>
<th>Send fraction</th>
<th>Increase factor</th>
<th>Increase $N'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>2.78</td>
<td>0.08</td>
<td>1.06</td>
<td>42</td>
</tr>
<tr>
<td>120</td>
<td>8.26</td>
<td>0.25</td>
<td>1.24</td>
<td>148</td>
</tr>
<tr>
<td>240</td>
<td>16.50</td>
<td>0.50</td>
<td>1.63</td>
<td>390</td>
</tr>
<tr>
<td>360</td>
<td>24.76</td>
<td>0.75</td>
<td>2.39</td>
<td>861</td>
</tr>
<tr>
<td>480</td>
<td>33.06</td>
<td>0.99</td>
<td>4.52</td>
<td>2170</td>
</tr>
</tbody>
</table>

Your Mileage May (Will) Vary
Summary

• Kernel-groupcast
  – OS interface and mechanism for group unicast
  – relative minor modifications to FreeBSD and Linux
  – significantly decrease time for group sends
• Does not reduce data sent
  – improves server efficiency (efficient group unicast)
• Main source of improvement not reduced mem copy
  – tight kernel loop
    • reduced interrupts
    • improved cache utilization
**Future Work**

- Detailed breakdown of network I/O cost components
  - better understanding
  - on a variety of hardware platforms
- Better models for expected scalability
- Variety of apps and interaction with kernel-groupcast
  - library to work with existing interfaces?
- Apply kernel-groupcast to other transport protocols