A Reliable and Secure Connection Migration Mechanism in Mobile Codes

Xiliang Zhong, Cheng-Zhong Xu, and Haiying Shen
Dept. of Electrical and Computer Engineering
Wayne State University, Detroit, Michigan
{xlzhong,czxu,shy}@wayne.edu
Outline

- Background and Goal
- Message delivery protocols
- NapletSocket: Sync. transient comm
- Experimental evaluation
- Conclusion
Code Mobility

- **Mobility**
  - Nature of human being, moving toward resources or away from scarcity
  - User/device mobility vs computation migration

- **Mobile agents:**
  - Autonomous object, acting behalf of its client
  - Able to travel proactively from machine to machine on open and distributed systems, carrying its code state.
Comm. in Multi-Agent Systems

- **Multi-Agent comm. languages**
  - KQML and FIPA’s ACL
- **Message delivery protocols**
  - Mailbox-based mechanism [Cao etc 02]
  - Reliable message delivery [Cao, et al 04, Murphy et al ’04]

**Persistent asynchronous**

Submitted messages are stored in the middleware system as long as it takes to deliver it.
Transient Synchronous Comm

- **Transient**: messages are delivered only if both the sender and receiver are executing.
  - E.g. socket
- Complement to persistent comm. in support of collaborative computing

How to deal with mobility?
- Connection migration
- Exactly-once delivery
Related works

• Network layer
  - Mobile IP (Ioannidis et al 2002)

• Transport layer
  - No control over the logical mobility

• Session layer
Related works (Conts’)

• **Session layer**
  - Connection migration for code mobility
    • Persistent Connection (Zhang, et al 1995)
    • Mobile TCP (Qu, et al 1997)
    • MobileSocket (Okoshi, et al 1999)
  - Fault tolerant connection
    • Robust TCP (Ekwall, et al 2002)
    • Reliable Sockets (Zandy, et al 2002)
  - No support for dual mobility of agents
Design of NapletSocket

- A library built on top of Java Socket
- Similar APIs to standard socket
  - `NapletSocket(agent-id)`, `NapletServerSocket(aid)`
  - suspend, resume
- Integrated into **Naplet** mobile agent system
Naplet System

- Naplet is a featured mobile agent system in support of composable adaptive network services
- First released in 1998. Latest release Naplet 0.19
  - Open source code: http://www.cic.eng.wayne.edu/software/naplet
  - Flexible: Separation of policy from mechanism
Naplet Features

- Structured navigation mechanism
  - Seq, Par, Alt, Loop
  - Regular-completeness
- Connection migration with mobile agents
- Proportional-share resource management
- Agent-oriented coordinated access control
Basic Ideas of NapletSocket

- Connection is suspended before migration and resumed after that.
- Data receiving and agent migration at the same time:
  - Undelivered data is buffered before migration and migrate with agent.
  - Delivered right after connection is resumed.
## State Transition

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSED</td>
<td>Not connected</td>
</tr>
<tr>
<td>LISTEN</td>
<td>Ready to accept connections</td>
</tr>
<tr>
<td>CONNECT_SENT</td>
<td>Sent a CONNECT request</td>
</tr>
<tr>
<td>CONNECT_ACKED</td>
<td>Confirmed a CONNECT request</td>
</tr>
<tr>
<td>ESTABLISHED</td>
<td>Normal state for data transfer</td>
</tr>
<tr>
<td>SUS_SENT</td>
<td>Sent a SUSPEND request</td>
</tr>
<tr>
<td>SUS_ACKED</td>
<td>Confirmed a SUSPEND request</td>
</tr>
<tr>
<td>SUSPEND_WAIT</td>
<td>Wait in a suspend operation</td>
</tr>
<tr>
<td>SUSPENDED</td>
<td>The connection is suspended</td>
</tr>
<tr>
<td>RES_SENT</td>
<td>Sent a RESUME request</td>
</tr>
<tr>
<td>RES_ACKED</td>
<td>Confirmed a RESUME request</td>
</tr>
<tr>
<td>RESUME_WAIT</td>
<td>Wait in a resume operation</td>
</tr>
<tr>
<td>CLOSED_SENT</td>
<td>Sent a CLOSE request</td>
</tr>
<tr>
<td>CLOSE_ACKED</td>
<td>Confirmed a CLOSE request</td>
</tr>
</tbody>
</table>
Open/Close
Suspend/Resume

C. XU@WSU
Concurrent Migration

- Basic idea is to let one of the agents migrate first
- The first one signals the blocked one after it finishes migration
- No need to resume the connection when the first one finishes migration
- Set priorities for the agents.
  - Statically assign priority to client or server is not deadlock free
  - Determine priority based on agent ID
Concurrent Suspend

C. XU@WSU
Concurrent Suspend (I)

- Occur at the same time.
  - Assume B has a higher priority
Concurrent Suspend (II)

- One issues suspend request before the other completes.
Security Concerns

- Only the one who established a connection can operate on it.
- By default, a request+Socket ID is enough for an action; no access to physical resource like port num
- Use of a secret session key
  - Diffie-Hellman key exchange protocol
  - Negotiate a key during connection setup and verify it during operation on the connection
  - Free from eavesdropping
Experimental Evaluation

• Experiments with NapletSocket vs. Java Socket
  – A group of Sun Blade workstation in 100Mbps Ethernet
  – Performance metrics
    • Cost of primitive operations
    • Throughput
Effectiveness of reliable comm.

- Exactly-Once Delivery Guarantee
## Cost of primitive ops

<table>
<thead>
<tr>
<th>Connection Type</th>
<th>Open (ms)</th>
<th>Close (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Socket</td>
<td>3.7</td>
<td>0.6</td>
</tr>
<tr>
<td>NapletSocket w/o security</td>
<td>18.2</td>
<td></td>
</tr>
<tr>
<td>NapletSocket w. security</td>
<td>134.4</td>
<td></td>
</tr>
</tbody>
</table>

![Graph showing the cost components of NapletSocket](image-url)
Cost of primitive ops (cont’)

• Suspend and Resume
  – 27.8ms for suspend and 16.9ms for resume
  – The latency for suspend operation mainly due to exchange of message and check for input data
  – For resume operation, the cost is mainly for exchange of message and update of data socket and I/O

• Benefit of provision a reliable connection
  – suspend+resume: 44.7ms
  – close+reopen: 138.6ms
Throughput between stationary Agents

![Graph showing throughput comparison between Java Socket and NapletSocket for different message sizes.](image-url)
Throughput of Itinerary Agents

- Both agents keep sending messages to each other while they are migrating
- **Effective throughput**: total traffic communicated in a time unit
Impact of migration frequency

![Bar Chart]

- **Throughput (Mb/sec)**

- **Agent service time (sec)**

- **With migration**
- **Without migration**

The chart compares the throughput with and without migration for different agent service times.
Impact of migration hops.

![Graph showing impact of migration hops](chart.png)
Conclusions

• **NapletSocket** for transient synchronous communication between mobile agents.
  - Overhead of NapletSocket is marginal for comm. between stationary agents.
  - Overhead of NapletSocket between itinerary agents is determined by their migration frequency.

• **Future work**
  - Fault-tolerant for host/network failure
  - Group communication
A Reliable and Secure Connection Migration Mechanism for Mobile Codes

Thanks You!