Deploying Safe User-Level Network Services with icTCP

Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
Computer Sciences Department
University of Wisconsin, Madison
OSDI ‘04
Outline

• Introduction
• icTCP Design
• Methodology
• Evaluation
• TCP Extensions
Introduction
Introduction (cont.)
Introduction (cont.)

• Address the problem of deployment
  – proposing a small but enabling change to the network stack.

• icTCP
  – Exports key pieces of state information.
  – Provides safe control to user-level libraries.
  – Deployable
  – Flexible
  – Composable
Introduction

- icTCP framework is easy to implement.
  - Linux-based implementation requires approximately 300 new lines of code.
  - Reduces the chances of introducing new bugs into the protocol.
- icTCP is the safe manner in which it provides new user-level control.
  - Users are allowed to control only a set of *limited* virtual TCP variables.
icTCP Design
icTCP Design (cont.)

- Different TCP connections can use different icTCP libraries.
- icTCP provides both a polling and interrupt-based interface.
- icTCP exports all variables that are part of the TCP specification.
icTCP Design (cont.)

- icTCP exposes standard information about each packet.
  - Message list.
  - ACK list.
  - List created only when enabled by user-level services.
- How icTCP allow variables that are internal to TCP to be externally set in a safe manner.
  - Limited virtual variable.
  - Restrict the range of value.
icTCP Design (cont.)

- icTCP accepts all settings and coerces the virtual variable into a valid range.

**Example**
- $0 \leq \text{vcwnd} \leq \text{cwnd}$
- If vcwnd rises above cwnd the value of cwnd is used instead.
### icTCP Design (cont.)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Safe Range</th>
<th>Example usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwnd</td>
<td>Congestion window</td>
<td>$0 \leq v \leq x$</td>
<td>Limit number of sent packets</td>
</tr>
<tr>
<td>cwnd.cnt</td>
<td>Linear cwnd increase</td>
<td>$0 \leq v \leq x$</td>
<td>Increase cwnd less aggressively</td>
</tr>
<tr>
<td>ssthresh</td>
<td>Slow start threshold</td>
<td>$1 \leq v \leq x$</td>
<td>Move to SS from CA</td>
</tr>
<tr>
<td>rcv.wnd</td>
<td>Receive window size</td>
<td>$0 \leq v \leq x$</td>
<td>Reject packet; limit sender</td>
</tr>
<tr>
<td>rcv.nxt</td>
<td>Next expected seq num</td>
<td>$x \leq v \leq x + vrcv.wnd$</td>
<td>Reject packet; limit sender</td>
</tr>
<tr>
<td>snd.nxt</td>
<td>Next seq num to send</td>
<td>vsnd.una $\leq v \leq x$</td>
<td>Reject ack; enter SS</td>
</tr>
<tr>
<td>snd.una</td>
<td>Oldest unacked seq num</td>
<td>$x \leq v \leq vsnd.nxt$</td>
<td>Reject ack; enter FRFR</td>
</tr>
<tr>
<td>dupthresh</td>
<td>Duplicate threshold</td>
<td>$1 \leq v \leq vewnd$</td>
<td>Enter FRFR</td>
</tr>
<tr>
<td>RTO</td>
<td>Retransmission timeout</td>
<td>$\text{exp.backoff} \times (\text{srtt} + \text{rttvar}) \leq v$</td>
<td>Enter SS</td>
</tr>
<tr>
<td>retransmits</td>
<td>Number of consecutive timeouts</td>
<td>$0 \leq v \leq \text{threshold}$</td>
<td>Postpone killing connection</td>
</tr>
</tbody>
</table>
### icTCP Design (cont.)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Safe Range</th>
<th>Example usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwnd</td>
<td>Congestion window</td>
<td>$0 \leq v \leq x$</td>
<td>Limit number of sent packets</td>
</tr>
<tr>
<td>cwnd.cnt</td>
<td>Linear cwnd increase</td>
<td>$0 \leq v \leq x$</td>
<td>Increase cwnd less aggressively</td>
</tr>
<tr>
<td>ssthresh</td>
<td>Slow start threshold</td>
<td>$1 \leq v \leq x$</td>
<td>Move to SS from CA</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Safe Range</td>
<td>Example usage</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
<td>-----------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>rcv.wnd</td>
<td>Receive window size</td>
<td>$0 \leq v \leq x$</td>
<td>Reject packet; limit sender</td>
</tr>
<tr>
<td>rcv.nxt</td>
<td>Next expected seq num</td>
<td>$x \leq v \leq x + vrcv.wnd$</td>
<td>Reject packet; limit sender</td>
</tr>
<tr>
<td>snd.nxt</td>
<td>Next seq num to send</td>
<td>$vsnd.una \leq v \leq x$</td>
<td>Reject ack; enter SS</td>
</tr>
<tr>
<td>snd.una</td>
<td>Oldest unackd seq num</td>
<td>$x \leq v \leq vsnd.nxt$</td>
<td>Reject ack; enter FRFR</td>
</tr>
</tbody>
</table>
# icTCP Design

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Safe Range</th>
<th>Example usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>dupthresh</td>
<td>Duplicate threshold</td>
<td>$1 \leq v \leq \text{vcwnd}$</td>
<td>Enter FRFR</td>
</tr>
<tr>
<td>RTO</td>
<td>Retransmission timeout</td>
<td>$\exp.\text{backoff} (\text{srtt} + \text{rttvar}) \leq v$</td>
<td>Enter SS</td>
</tr>
<tr>
<td>retransmits</td>
<td>Number of consecutive timeouts</td>
<td>$0 \leq v \leq \text{threshold}$</td>
<td>Postpone killing connection</td>
</tr>
</tbody>
</table>
Methodology
Methodology

- icTCP is implemented in the Linux 2.4.18 kernel.
- Experiments are performed exclusively within the Netbed network emulation environment.
- A single Netbed machine contains
  - 850 MHz Pentium 3 CPU
  - 512MB of main memory
  - five Intel EtherExpress Pro 100Mb/s Ethernet ports.
Evaluation
<table>
<thead>
<tr>
<th>Information</th>
<th>LOC</th>
<th>Control</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>25</td>
<td>cwnd</td>
<td>15</td>
</tr>
<tr>
<td>Message List</td>
<td>33</td>
<td>dupthresh</td>
<td>28</td>
</tr>
<tr>
<td>Ack List</td>
<td>41</td>
<td>RTO</td>
<td>13</td>
</tr>
<tr>
<td>High-resolution RTT</td>
<td>12</td>
<td>ssthresh</td>
<td>19</td>
</tr>
<tr>
<td>Wakeup events</td>
<td>50</td>
<td>cwnd_cnt</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>retransmits</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rcv_nxt</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rcvWnd</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>snd_una</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>snd_nxt</td>
<td>14</td>
</tr>
<tr>
<td><strong>Info Total</strong></td>
<td>161</td>
<td><strong>Control Total</strong></td>
<td>155</td>
</tr>
<tr>
<td><strong>icTCP Total</strong></td>
<td></td>
<td><strong>316</strong></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation (cont.)

```c
// set internal TCP variables
tcp_setsockopt (option, val) {
    switch (option) {
      case TCP_USE_VCWND:
        use_vcwnd = val;
      break;
      case TCP_SET_VCWND:
        vcwnd = val;
      break;
    }
}

// check if data should be put on the wire
tcp_snd_test () {
    if (use_vcwnd)
        min_cwnd = min (vcwnd, cwnd);
    else
        min_cwnd = cwnd;
    // if okay to transmit
    if (((tcp_packets_in_flight < min_cwnd) &&
        /* ... other rules .... */)
        return 1;
    else
        return 0;
}
Evaluation (cont.)
Evaluation (cont.)

**dupthresh**
- $3 < \text{dupthresh} < 43$ (U)
- $\text{dupthresh} < 3$ (U)

**RTO**
- $\text{srtt + var}$ (D)
- $\text{srtt + var}$ (U)
Evaluation (cont.)

![Scaling - CPU](image1)

![Scaling - Throughput](image2)
Evaluation (cont.)

- icTCP-Vegas
  - Carefully matching the sending rate to the rate at which packets are being drained by the network.

- Implementation
  - Placed in a user-level library.
  - Library simply passes all messages directly to icTCP
  - Poll icTCP for new information
    - Send a new packet
    - An acknowledgment is received
    - A round ends.
Evaluation

- Gets the relevant TCP state.
- Calculates target congestion windows, vcwnd.
- icTCP-Vegas sets this value explicitly inside icTCP.

- Evaluation

![Graph showing CPU Utilization vs Bandwidth (Mb/s) - no delay with different TCP variants]
TCP Extensions (cont.)

- icTCP-Nice
  - Provides a near zero-cost background transfer.
  - Reaps a large fraction of the spare network bandwidth.
  - $\text{minRTT} + (\text{maxRTT} - \text{minRTT}) \times t$ (where $t = 0.1$)
  - Allows the windows to be less than one.
TCP Extensions (cont.)

- Implementation
  - Obtains the full message list containing the sequence number and round trip time of each packet.
  - For window of 1/n
    - Sets vcwnd to 1 for single RTT period
    - Sets to 0 for (n-1) periods
TCP Extensions (cont.)

![Graph showing Link Capacity Vs Latency](image)

- Reno
- TCP Nice
- icTCP-Nice

- X-axis: Bottleneck Link Bandwidth (Mbps)
- Y-axis: Document Latency (ms)

The graph illustrates the relationship between link capacity and latency for different TCP extensions, demonstrating how latency decreases as bandwidth increases.
TCP Extensions (cont.)
TCP Extensions (cont.)

- icCM
  - Applications that employ multiple concurrent flows that compete with each other for resources and do not share network information with each other.
  - UDP-based flows without sound congestion control.
  - CM inserting a module above IP at both the sender and receiver.
    - Maintains network statistics across flows.
TCP Extensions (cont.)

• Implementation
  – icCM server has two roles
    • To identify macroflows.
    • Track the aggregate statistics associated with each macroflow.
  – icCM clients obtain from the server
    • the number of flows in this macroflow.
    • the total number of outstanding bytes in this flow
TCP Extensions (cont.)
• icTCP-RR
  – Packet reordering may be more common in the internet than earlier designers suspected.
  – Sender receives a rash of duplicate ACKs and wrongly concludes that a loss has occurred.
  – Implementation base on Blanton and Allman’s work.
    • Limits the maximum value of dupthresh to 90% of the window size.
    • When timeout occurs, sets dupthresh back to its original value of 3.
TCP Extensions (cont.)

• Implementation
  – Keeps a history of ACKs received.
  – When DSACK arrives, icTCP places the sequence number of the falsely retransmitted packet into the ACK list.
  – Library searches through past history to measure the reordering length and sets dupthresh accordingly.
TCP Extensions (cont.)

False Fast Retransmits

Throughput

Packet Delay Rate (%)

Packet Delay Rate (%)
TCP Extensions (cont.)

- icTCP-EFR
  - In wireless LANS
    - Loss is much more common
    - Duplicate acks should be used a strong signal of packet loss.
    - The value of dupthresh should be lowered.
TCP Extensions (cont.)

• Implementation
  – When the window is small, the library frequently checks the message list for duplicate acks.
  – When it sees one, it computes and sets a new value.
TCP Extensions (cont.)

![Graphs showing retransmissions and throughput over loss rate for different TCP extensions.]
TCP Extensions (cont.)

• icTCP-Eifel
  – Correctly setting RTO can greatly influence performance
  – If RTO expire prematurely
    • Forcing unnecessary spurious.
  – An overly-conservative RTO
    • Cause long idle times.
TCP Extensions (cont.)

Figure 3: Prediction Flaw in $RTO_L$.

Figure 4: A Collapsed $RTO_E$ (model).
TCP Extensions (cont.)

• Implementation
  – Access three icTCP variables
    • mRTT
    • ssthresh
    • cwnd
  – It wakes when an acknowledgment arrives.
  – Poll icTCP for the new mRTT.
TCP Extensions
The End