Engineering Fault-Tolerant TCP/IP servers using FT-TCP

Dmitrii Zagorodnov

University of California San Diego
Motivation

- Reliable network services are desirable
- ...but costly!
  - Extra and/or specialized hardware
  - Custom software, deployment
  - Performance

- Guiding principles:
  - Off-the-shelf components
  - No changes to the applications, protocols, and OS
  - Minimize interference with execution and data flow
Problem
State Coordination

- Theory is well-understood
  - Same inputs \(\rightarrow\) Same behavior
  - Use active backup, primary backup, rollback recovery...

- Applying it is hard
  - No simple, well-defined boundary
    - system calls
    - signals
    - shared memory
Client-assisted failover

- Very popular [Veritas, IBM, etc.]
- Problems:
  - Engineering cost
  - Redoing work
  - Deployment
- Redundant hardware is common
Socket-level Approach

- Provide a **reliable socket** by reconnecting...
  - in the middleware [Orgiyan Fetzer ‘01] [Nasika Dasgupta ‘00]
  - in the protocol [Snoeren et al ‘01] [Sultan Srinivasan Iftode ‘01]

- Advantage: Allows failover to a distant replica
- Problem: Requires changing software on clients
Proxy-based Approach

- Use a **proxy** on the network path
- Advantage: Client is unchanged
- Problems:
  - Single point of failure
  - Potential performance bottleneck
Server-Side Approach

- Transfer TCP and application state
  - [INFOCOM ‘01], a separate session at DSN ‘03
- Advantages:
  - changes only to server
  - no single point of failure
- Problem: TCP implementation-specific
Our Approach: FT-TCP

- "Wrap" the server’s TCP/IP stack

- Advantages:
  - No changes to TCP
  - No changes to client
  - No single point of failure
Talk Outline

- Failure-free FT-TCP operation
- Failure-free performance
- State Synchronization
- Handling Failures
TCP Basics

Retransmission

Timeout

A

B

A4
A3
A2
A1

A4
A3
A2
A1

A1
A2
A3

ack A1
win 2

ack A3
win 0

ack A3
win 3

ack A4
win 2

A4
A4
A4

A4
A4
A4

A1
A1
A2
A2
A2
A3

A1
A1
A1
A1

May 19, 2004

Engineering Fault-Tolerant TCP/IP Servers using FT-TCP

11
FT-TCP Architecture

- Cold & Hot backups
- They differ in timing, not functionality
- Primary mode & Backup mode
Primary Mode

```
recv( ), size
ack
send( )
recv( ), size
ack
send( )
```

Primary Server

NSW

TCP/IP

SSW

Stable Buffer

May 19, 2004
Engineering Fault-Tolerant TCP/IP Servers using FT-TCP
Backup Mode

- Create a backup process with a connection

**Backup**

```
accept()  
```

```
ok
```

**NSW**

```
accept()  
```

```
ok
```

**TCP/IP**

```
syn
```

```
syn+ack
```

```
ack
```

**SSW**

- Backup process runs using information from the stable buffer

**Backup**

```
read()  
```

```
data, ret
```

**NSW**

- Upon failure, exhaust the buffer & insert backup for primary
TCP state synchronization

- States of TCP stacks on replicas are different
  - Initial Sequence Number
    - Primary (ISN=10)
      - 10 11 12
    - Backup (ISN=20)
      - 20 21 22

- IP address
  - Translate address in IP headers

- Incremental checksum is fast
Talk Outline

- Failure-free FT-TCP operation
- Failure-free performance
- State Synchronization
- Handling Failures
Performance Measurements

- Metrics:
  - Failure-free throughput
  - Failure-free latency
  - Failover duration

- Experiments:
  - Bulk transfers (4-40MB)
  - Small requests
  - Induce primary failure
Applications

- Micro-benchmark (ttcp)
- Samba
  - Long-lived TCP connections
  - SMB clients give up aborted data transfers
- Darwin Streaming Server (DSS)
  - RTSP connections over TCP port 80 are common
  - QuickTime player pauses when connection breaks
- Apache
  - Popular
## Micro-benchmark throughput

<table>
<thead>
<tr>
<th>Incoming Transfer</th>
<th>Shared</th>
<th>10</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client - Server Link Bandwidth (Mbps)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>43%</td>
<td>86%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>85%</td>
<td>100%</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td>60%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outgoing Transfer</th>
<th>Shared</th>
<th>10</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client - Server Link Bandwidth (Mbps)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>97</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
### Samba throughput

<table>
<thead>
<tr>
<th></th>
<th>Primary - Backup Link Bandwidth (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Shared</td>
</tr>
<tr>
<td><strong>Incoming</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Transfer</strong></td>
<td></td>
</tr>
<tr>
<td>Client - Server</td>
<td></td>
</tr>
<tr>
<td>Link Bandwidth</td>
<td></td>
</tr>
<tr>
<td>(Mbps)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>38</td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td><strong>Outgoing</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Transfer</strong></td>
<td></td>
</tr>
<tr>
<td>Client - Server</td>
<td></td>
</tr>
<tr>
<td>Link Bandwidth</td>
<td></td>
</tr>
<tr>
<td>(Mbps)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>
Failure-free Performance

- Throughput under 100-1000 configuration:
  - Samba: over 95% of Clean throughput
  - DSS & Apache: ~100%
- Tuning for best results
  - Nagle algorithm, acknowledgement approach
- Scales with the number of connections (up to 128)
- Latency:
  - Insignificant for bulk transfers
  - Noticeable for small requests
Talk Outline

- Failure-free FT-TCP operation
- Failure-free performance
- State Synchronization
- Handling Failures
Application state synchronization

- Consistent network input is not sufficient
- Conservative approach: “Virtual Process Environment”
  - Performance overhead
  - Engineering challenge
- Minimal interception
  - Theoretically, hard
  - Empirically...
Non-Determinism

- System-induced ND
  - system calls, threads, signals

- Application-induced ND: random number
  - QuickTime: in the form of a random SessionID
  - Samba:
    - File handle
    - Random challenge used in authentication
  - ND is part of these protocols
Consistency filter

- Allow inconsistent system call results
- NSW consistency filter based on the protocol specs

- Generalizes SSW packet manipulations
Talk Outline

- Failure-free FT-TCP operation
- Failure-free performance
- State Synchronization
- Handling Failures
Handling Failures

- Detect failures
  - process crash: close( ) on the socket
  - machine crash: heartbeat timeout
- Maintain connection
  - with “closed window” acks
- Roll forward the backup process
  - Cold: start a process, exhaust the buffer
  - Hot: exhaust the buffer
- Ethernet address switch
- Restart the flow of data
End-point switch

- Ethernet address differences:
  - Hub & promiscuous mode
  - Use broadcast Ethernet address & virtual NIC
  - Update gateway’s IP → Ethernet mapping via gratuitous ARP
  - Change Ethernet address of the backup

- IP address differences:
  - Everyone has unique IP addresses & translates IP in headers
  - Everyone has the same IP address via an alias
Restart the flow of data

![Graph showing client and server activity over time]

- Client sent: Line graph with points
- Server acknowledged: Scatter plot with crosses

- Time (sec) range: 1.00 to 4.50
- Relative sequence numbers x 10^-6 range: 0.75 to 0.82

May 19, 2004
Failover Latencies

- Failure Detection: 40 ms
- Process Startup: 10 ms
- Replay (Minimal): 35 ms
- Replay (Samba Bulk): 160 ms
- Replay (Samba Small): 8000 ms
- Endpoint Switch: 2 ms
- Retransmission (Short): 200 ms
- Retransmission (Long): 64000 ms

Duration (Milliseconds) per 1MB
Conclusions

- **FT-TCP**: low-overhead, scalable, connection failover
  - Failure-free throughput overhead under 5%
  - Latency
    - unaffected for bulk transfers
    - noticeable for small requests
  - Hot backup is almost as fast as cold backup

- **Recovery**:  
  - Hot backup can recover in 200ms or less

- **Few sources of non-determinism in applications**  
  - Application-level consistency filter
Thank you!

Questions?
TCP Packets

- Initial Sequence Number
- Length
- Acknowledgement Number
- Sent, Acked
- Sent, Not Acked
- Can send
- Cannot send
- Advertised Window

TCP Packets diagram showing sequence number, length, acknowledgement number, and various states such as sent, acknowledged, not acknowledged, and ability to send.
Primary Mode

client

SSW

server

TCP

\[ sn' + \Delta_o \]
\[ stable_{seq} + \Delta_i \]
\[ w' + (asn' - stable_{seq}) \]

\[ sn = \text{stable}_{seq} + \Delta_o \]
\[ asn = \text{stable}_{seq} + \Delta_o \]

\[ sn' = \text{stable}_{seq} + \Delta_i + w' + (asn' - \text{stable}_{seq}) \]

\[ stable_{seq} = sn + L \]
Backup Mode

SSW

seq : 100
ack :

TCP

SYN
SYN+ACK
ACK

NSW

promotion

stable_seq: 300
primary_seq: 400
\[ \Delta_i = 300 - 100 - 1 \]
\[ \Delta_0 = 400 - 200 - 1 \]

\[
\begin{align*}
\text{stable_seq: } & 300 \\
\text{primary_seq: } & 400 \\
\Delta_i & = 300 - 100 - 1 \\
\Delta_0 & = 400 - 200 - 1
\end{align*}
\]

system calls

write(10)

promotion

May 19, 2004
Engineering Fault-Tolerant TCP/IP Servers using FT-TCP
Interference with data flow

- Stable Buffer connection
  - Nagle algorithm
- Ack's to the client
  - Delayed - delay empty ack packets
  - Eager - ack for every stable packet
TCP Acknowledgements Problem

TCP Acknowledgements Problem

TCP
SSW
client
retransmission timeout

May 19, 2004
Engineering Fault-Tolerant TCP/IP Servers using FT-TCP
Delayed Ack Strategy
Eager Ack Strategy

**Diagram: Eager Ack Strategy**

- **client**
- **SSW**
- **TCP**

Numbers along the diagram represent the sequence of events in the Eager Ack strategy.
## Incoming ttcp

<table>
<thead>
<tr>
<th></th>
<th>Shared</th>
<th>10</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nagle</td>
<td>No Nagle</td>
<td>Nagle</td>
<td>No Nagle</td>
</tr>
<tr>
<td>Delayed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>43</td>
<td>38</td>
<td>86</td>
<td>77</td>
</tr>
<tr>
<td>Eager</td>
<td>43</td>
<td>38</td>
<td>86</td>
<td>77</td>
</tr>
<tr>
<td>Delayed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eager</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delayed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eager</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Incoming Samba

<table>
<thead>
<tr>
<th></th>
<th>Shared</th>
<th></th>
<th>10</th>
<th></th>
<th>100</th>
<th></th>
<th>1000</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nagle</td>
<td>No Nagle</td>
<td>Nagle</td>
<td>No Nagle</td>
<td>Nagle</td>
<td>No Nagle</td>
<td>Nagle</td>
<td>No Nagle</td>
</tr>
<tr>
<td>Delayed 10</td>
<td>21</td>
<td>38</td>
<td>55</td>
<td>75</td>
<td>61</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eager 10</td>
<td>34</td>
<td>37</td>
<td>55</td>
<td>75</td>
<td>61</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delayed 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td>78</td>
<td>15</td>
<td>95</td>
</tr>
<tr>
<td>Eager 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td>79</td>
<td>15</td>
<td>95</td>
</tr>
<tr>
<td>Delayed 1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>38</td>
</tr>
<tr>
<td>Eager 1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>63</td>
</tr>
</tbody>
</table>
### Outgoing Samba

<table>
<thead>
<tr>
<th></th>
<th>Shared</th>
<th>10</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nagle</td>
<td>No Nagle</td>
<td>Nagle</td>
<td>No Nagle</td>
</tr>
<tr>
<td><strong>Delayed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>10</strong></td>
<td>56</td>
<td>80</td>
<td>42</td>
<td>98</td>
</tr>
<tr>
<td><strong>Eager</strong></td>
<td>56</td>
<td>80</td>
<td>42</td>
<td>98</td>
</tr>
<tr>
<td><strong>Delayed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>100</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Eager</strong></td>
<td></td>
<td></td>
<td>8</td>
<td>92</td>
</tr>
<tr>
<td><strong>Delayed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1000</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Eager</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Latency of Samba Requests

- Samba Requests: Clean 2181, FT-TCP 6183
- TCP Packet: Clean 748, FT-TCP 2337
- Buffering: Clean 551

Average Latency (microseconds)
Challenge-response protocol

- Used for authentication in Samba and others

Server

\(\text{\{challenge\}} \text{hash(valid password)}\)

Client

\(\text{\{challenge\}} \text{hash(typed password)}\)

- Problem: filter cannot obtain client’s password or hash
- Filter must authenticate the client itself, and then force the server to make the same decision
  - If client is OK: send the hash of the correct password
  - If client is not OK: send anything but the correct hash
- Kerberos, SSL are possible
Other Crypto Protocols

- Kerberos
  - no randomness on the server

- Secure Socket Layer
  - random numbers on the client and on the server
  - given server’s private key filter can be the “man-in-the-middle”

- Limitations of the consistency filter
  - results of one-way functions cannot be retrieved and changed
    - e.g. anything encrypted with client’s public key
Recovery: 24 connections, 24ms break, retransmissions
Snooping Approaches

- Permanent snooping
  - No retransmission gap - minimal failover time
  - Heavy load on the CPU

- Reactive snooping
  - Start snooping upon failure
  - Catches the first retransmission - limits the gap to 200ms
  - When promotion is long (cold backup)