Increasing Availability of Linux System using Redundancy

Beob Kyun Kim (kyun@etri.re.kr)

Embedded SW Platform Research Division

2013-01-10

ETRI

Outline

I. Availability ?

II. Efforts to increase availability using redundancy

- History on Virtual Routers
- UCARP
- FTTCP and its variations

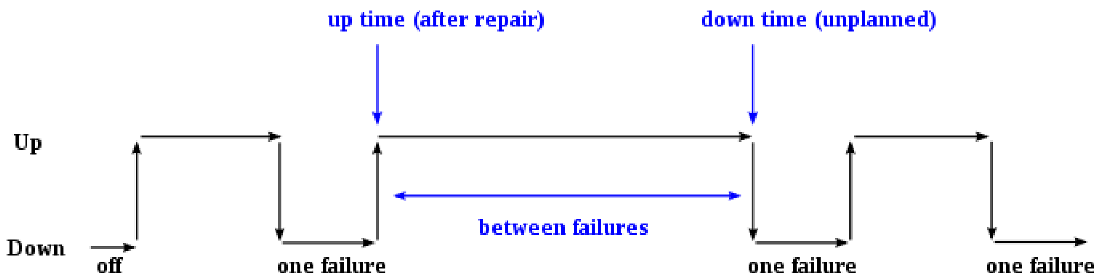
III. KSYNCD being developed by ETRI

Availability ?

Availability ?

- From wikipedia
  - is the proportion of time a system is in a functioning condition.
    - This is often described as a **mission capable rate**.
    - Mathematically, this is expressed as “ 1 - unavailability ”
  - MTTF, MTBF, MTTR, …

$$A = \frac{E[\text{Uptime}]}{E[\text{Uptime}] + E[\text{Downtime}]}$$



Time Between Failures = { down time - up time}

$$\text{Mean time between failures} = \text{MTBF} = \frac{\sum(\text{start of downtime} - \text{start of uptime})}{\text{number of failures}}.$$

Equation “A” from http://en.wikipedia.org/wiki/Availability

Equation & Diagram “MTBF” from http://en.wikipedia.org/wiki/Mean_time_to_failure

Availability in digits

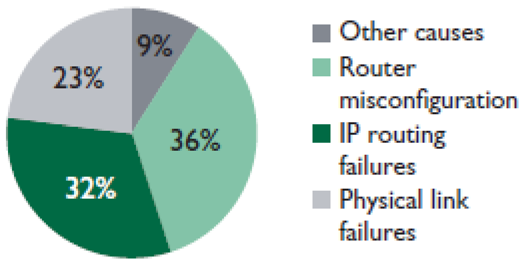
Availability %	Downtime per year	Downtime per month*	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% ("seven nines")	3.15 seconds	0.259 seconds	0.0605 seconds

from http://en.wikipedia.org/wiki/High_Availability

Availability in your life

- What 5-nines availability means in your life ?
  - The wired telephone infrastructure is engineered to guarantee 5-nines availability
    - *Routers for the Cloud, IEEE Internet Computing, 2011*
  - ➔ no interruption if no natural disaster

- Router interface downtime averaged roughly 955 minutes per year
  - one-year reliability study of IP core routers
    - *Internet Routing Instability, IEEE/ACM Trans. Networking, vol.6, no.5, 1998*
  - Doesn't reach 3-nines availability level (8.76 hours downtime per year)



What if ...

What if this router goes out?

What if a disk dies in this database server?

What if I unplug this networking cable?

What if this whole datacenter goes down?

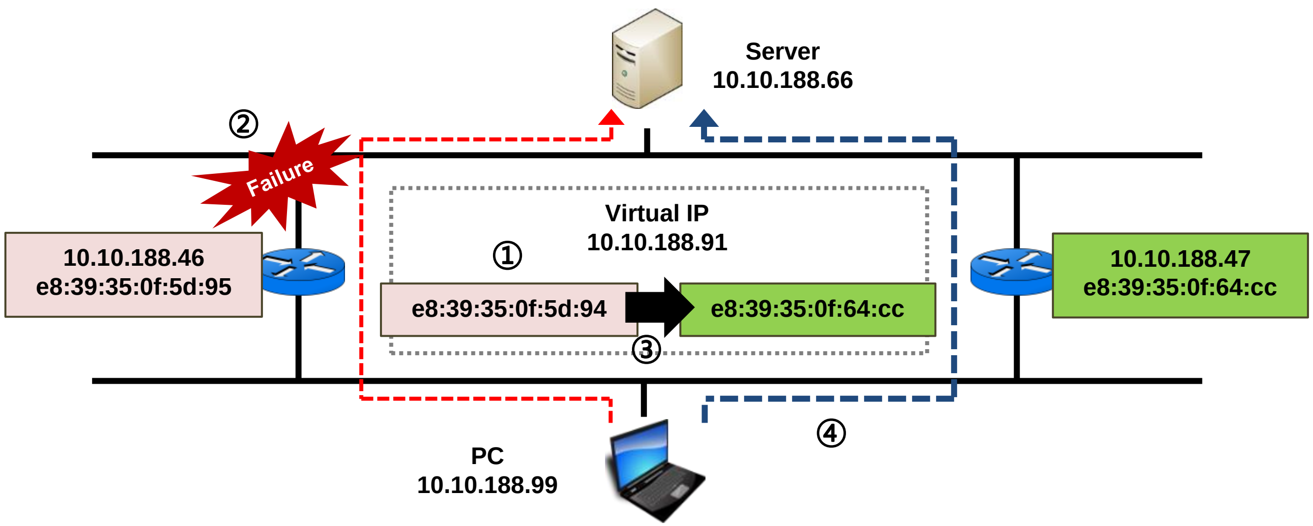
But, The show must go on ...

- History on Virtual Routers
- Introduction to UCARP and tests
- FTTCP and its variations

Efforts to increase Availability using Redundancy

Virtual Routers

- Concept
  - To allow hosts to appear to use a single router and to maintain connectivity even if the actual first hop router they are using fails
  - ← replace "router" with "server" or "service"



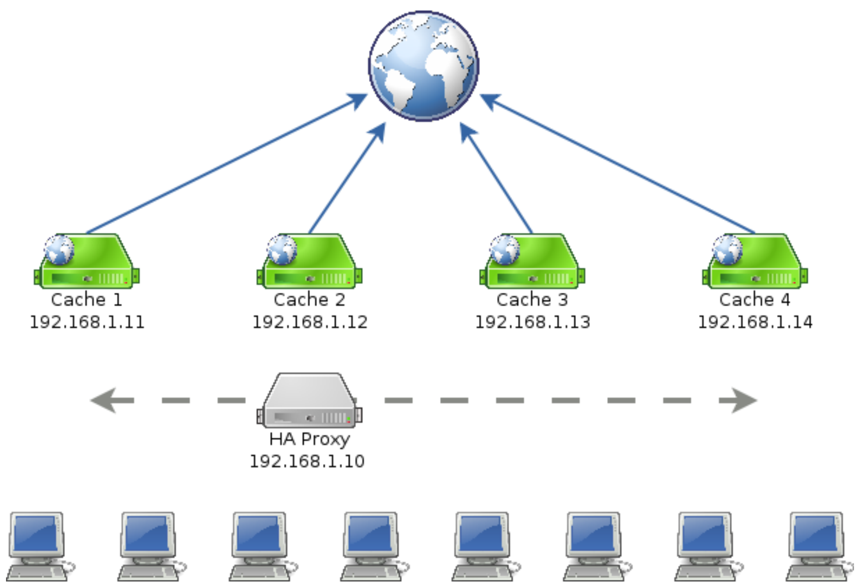
History on Virtual Routers

- 1990s, IETF began working on VRRP
  - VRRP : Virtual Router Redundancy Protocol
- From 1997, a debate between BSD developers and Cisco for the Intellectual Property of VRRP & HSRP (Cisco)
  - ➔ Cisco win
- OpenBSD dev.s started CARP as an alternative to the patented VRRP
  - CARP : Common Address Redundancy Protocol
  - Designed with security in mind
  - From Oct. 2003, became completely for free
- From May 2005, in FreeBSD 5.4
- Very complex status & history in Standard (IETF)
  - No official internet protocol number
- UCARP : a portable userland implementation of CARP by Frank DENIS

Characteristics of UCARP

- Strong points of UCARP (from ucarp.org)
  - Low overhead
  - Cryptographically signed messages
  - Interoperability between different OS
  - No need for any dedicated extra network link between redundant hosts
- Patent free implementation
- But the minimum heartbeat exchange interval is 1 second
  ➔ fault detection will be more later than 1 second

Deployment example



- Example of UCARP deploy.
  - The workstations all connect to the HAProxy instance at 192.168.1.10.
  - 192.168.1.10 is a *virtual* IP controlled by UCARP
  - HAProxy runs on *one* of the web cache servers at any given time, but any of the web caches can be the HAProxy instance.

Master election process

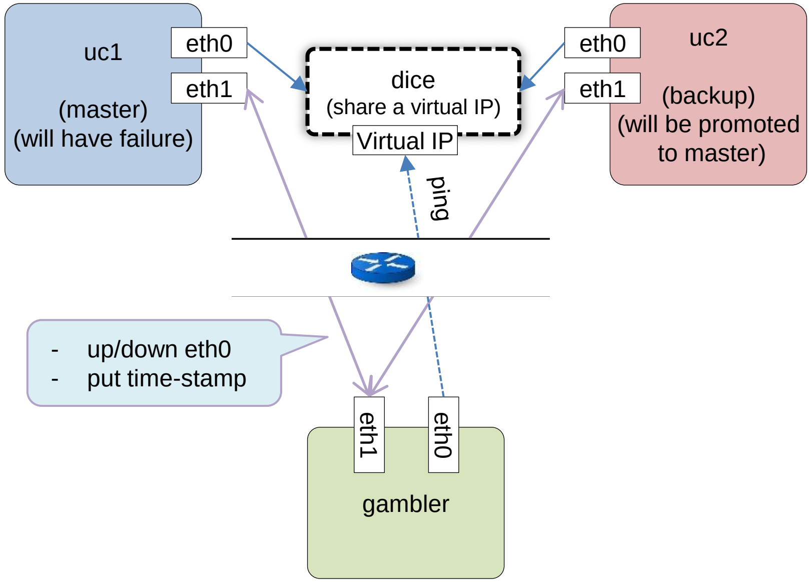
- A backup will become master if:
  - no one else advertises for 3 times its own advertisement interval
    - “deadratio”
  - you specified --preempt and it hears a master with a longer interval

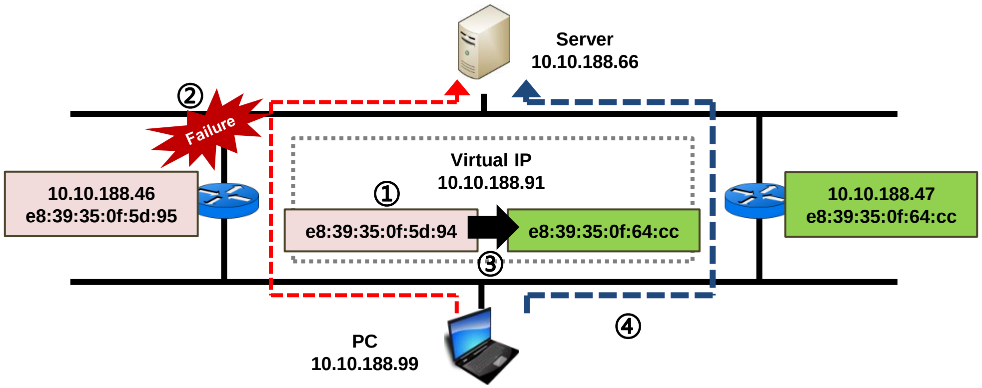
- A master will become backup if:
  - another master advertises a shorter interval
  - another master advertises the same interval, and has a lower IP address

- Factors that decide the possibility of being a master
  - Advertisement interval of each node
  - Preemptive mode of a node
  - Lower / higher IP address of each node

Test-bed

- ❖ 2 hosts for ucarp group
  - 2 GbE for each
  - 2 IP addresses for each
  - 1 shared virtual IP address
- ❖ 1 client host
- ❖ Switch for interconnection
  - L3 enabled
  - dedicated 1 Gbps ports x 16
  - Isolated network





Configuration and commands

- Configuration to attach UCARP to the physical eth.

```
auto eth0
iface eth0 inet static
        address 192.168.188.46
        netmask 255.255.255.0
        gateway 192.168.188.4

        up /usr/sbin/ucarp -i eth0 -s 192.168.188.46 -v 1 -p story -a 192.168.188.91 -u
/usr/share/ucarp/vip-up -d /usr/share/ucarp/vip-down -r 1 -z -B -M -n

iface eth0:ucarp inet static
        address 192.168.188.91
        netmask 255.255.255.0

auto eth1
iface eth1 inet static
        address 192.168.189.46
        netmask 255.255.255.0
```

Test and Results

- Current Test Status
  - Using binary distribution
  - In isolated network, 12~13 seconds for full recovery
    - Seems,
      most of consumed time is for DNS things while attaching Virtual IP to NIC
  - With connection to public network, 0.6 ~ 2.66 seconds for full recovery

Before "ifup eth0:ucarp"
Checked by ub2 (ucarp)
Failure of ub1 detected

After "ifdown eth0"
Checked by ub1 (ucarp)

```
=== 121211_181613 ================================
              1355217373.745155332 BEFORE_DOWN_ub1
ub2_UP_BEFORE 1355217374.597435735
ub1_DOWN_____ 1355217374.602441010
              1355217386.360554427 AFTER__DOWN_ub1
ub2_UP__AFTER 1355217386.898953368
```

Before "ifdown eth0"
Checked by gambler (client)

After "ifup eth0:ucarp"
Checked by ub2 (ucarp)
Around 12.33 sec. consumed

After "ifdown eth0"
Checked by gambler (client)
Around 12.6 sec. consumed

Test and Results

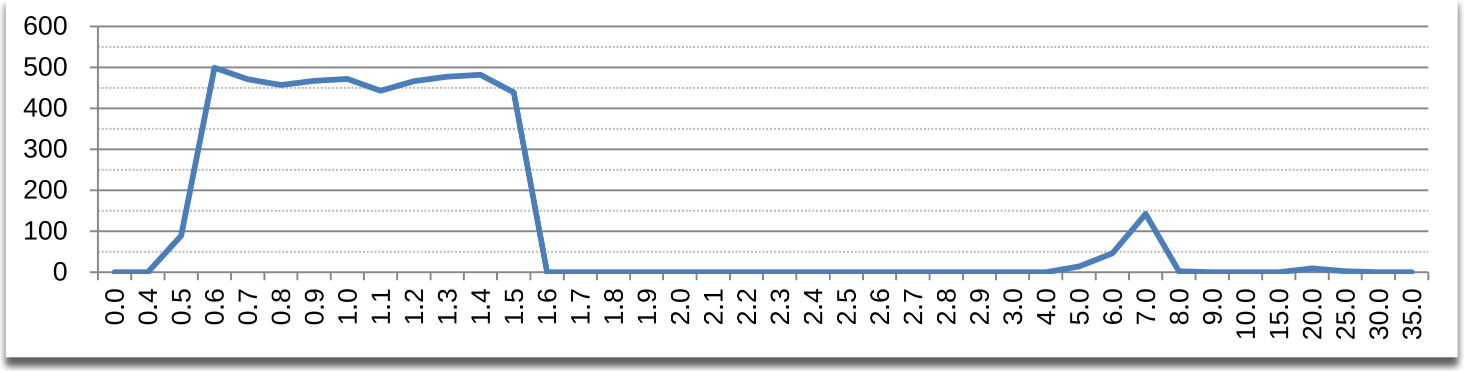
- Current Test Status

	Missing Try	Average	Max	Min	Variance	Std. Dev.
Raw	23	1.23948	24.0805	0.421579	1.84499	1.3583
Trimmed	-	1.00821	1.47741	0.536595	0.0765628	0.2767

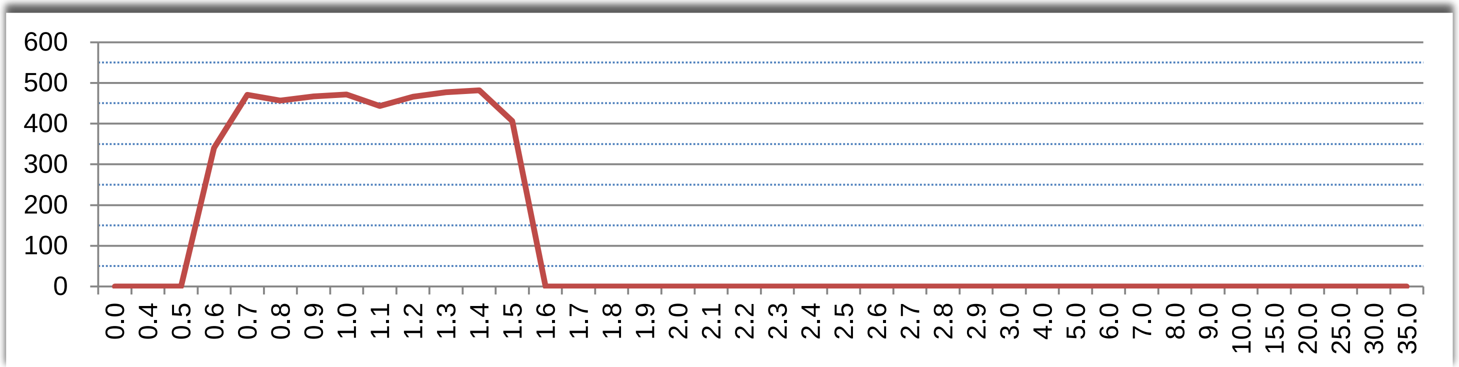
- Raw
  - Total 5,000 tries
  - 23 tries are missing ← too slow or too fast
- Trimmed
  - Cut off fastest 5% & slowest 5% from “Raw”

Test and Results

❖ Analysis



- Doesn't follow normal distribution
- Right side seems a kind of abnormal results (around 4.32%)



FTTCP

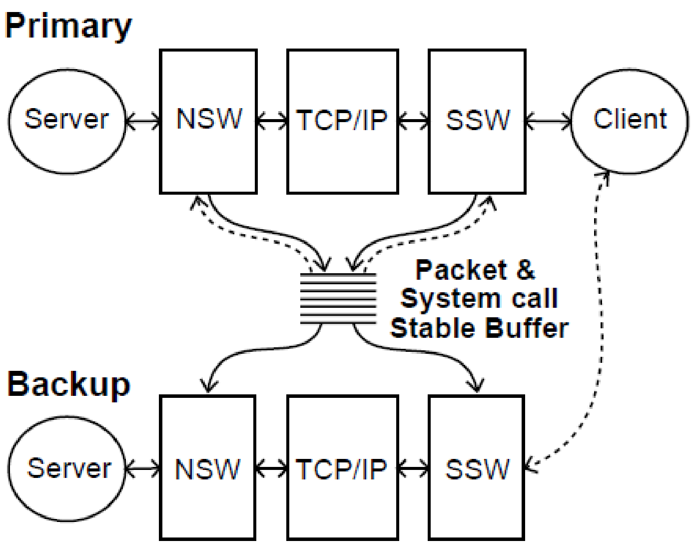
- Fault-Tolerant TCP
  - "Engineering Fault-tolerant TCP/IP servers using FT-TCP", D. Zagorodnov et al, DSN2003
- Does not require modifications to the TCP and does not affect any of the software running on the clients
  - But from our research, it's so hard and seems required to give a small modification to TCP stack to implement
- There're many researches which share the basic idea with FT-TCP
- But still difficult to implement a deployable version

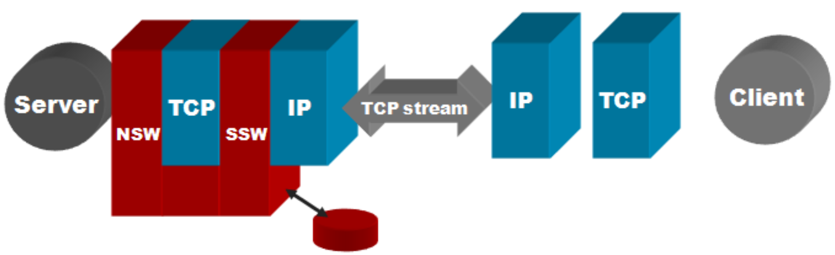
Related works

- Application-level recovery
  - The client app. Attempts to reestablish broken connections
  - Ex) FTP client, NFS, Samba
- Socket-level recovery
  - The failure is hidden by some lower layer that
    - Re-establishes connection and Provides a reliable socket to the application
  - Ex) wrapping standard library routines (socket, C, etc)
  - Requires upgrading some of the infrastructure (OS, protocol stack, or middleware) on the client host
- Server-side recovery
  - Restricts the fault-tolerance logic to the server cluster
  - Ex) FT-TCP

Architecture

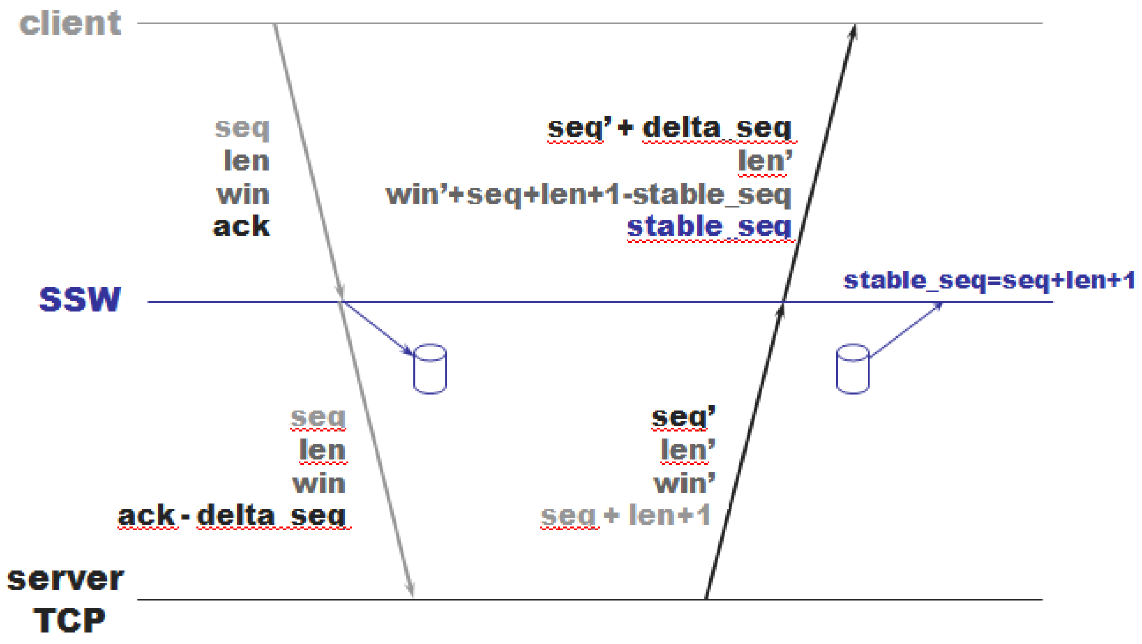
- FT-TCP is implemented by "wrapping" the TCP/IP stack
  - Intercept, modify, and discard packets on their way in and out of the TCP/IP stack using a component we call the SSW





Operations Example of FTTCP

- SSW fakes seq & ack numbers



Abstract Recovery Procedure

Abstract procedure of Fail-over

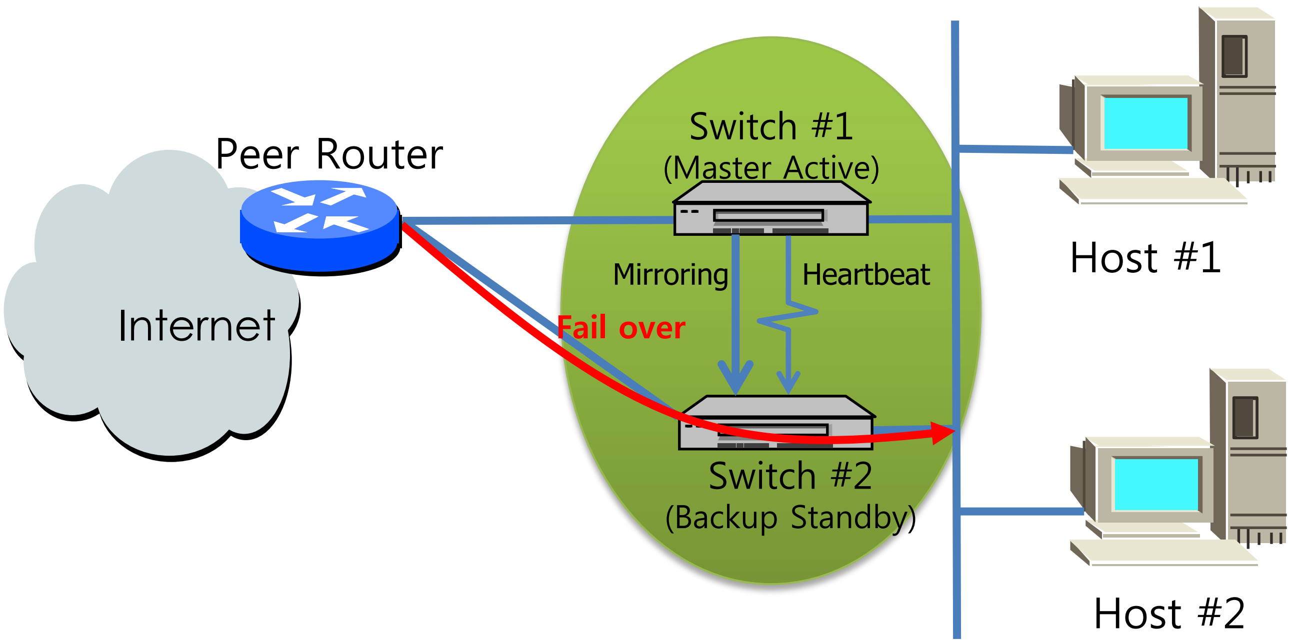
Failure detection	Promotion (master → backup)	Retransmission
- Backup checks heartbeats from master - No heartbeat means the failure of master	- Backup takes over roles from master while keeping TCP session	- Backup restarts the flow of data on the connection

KSYNCD FROM ETRI

Application to Carrier Ethernet System

Provide a base function of highly available Non-Stop active Router

- Failover using hardware redundancy



Application to Embedded Devices

There are cases that the system don't know while Kernel generates events in embedded linux kernel

Unlimited number of process generation in Android
↓
Low Performance
↓
Slow response against user's touch
↓
User starts to doubts of system failure
↓
Repeated errors
↓
User starts to consider other devices to replace

Vulnerabilities of Embedded Linux

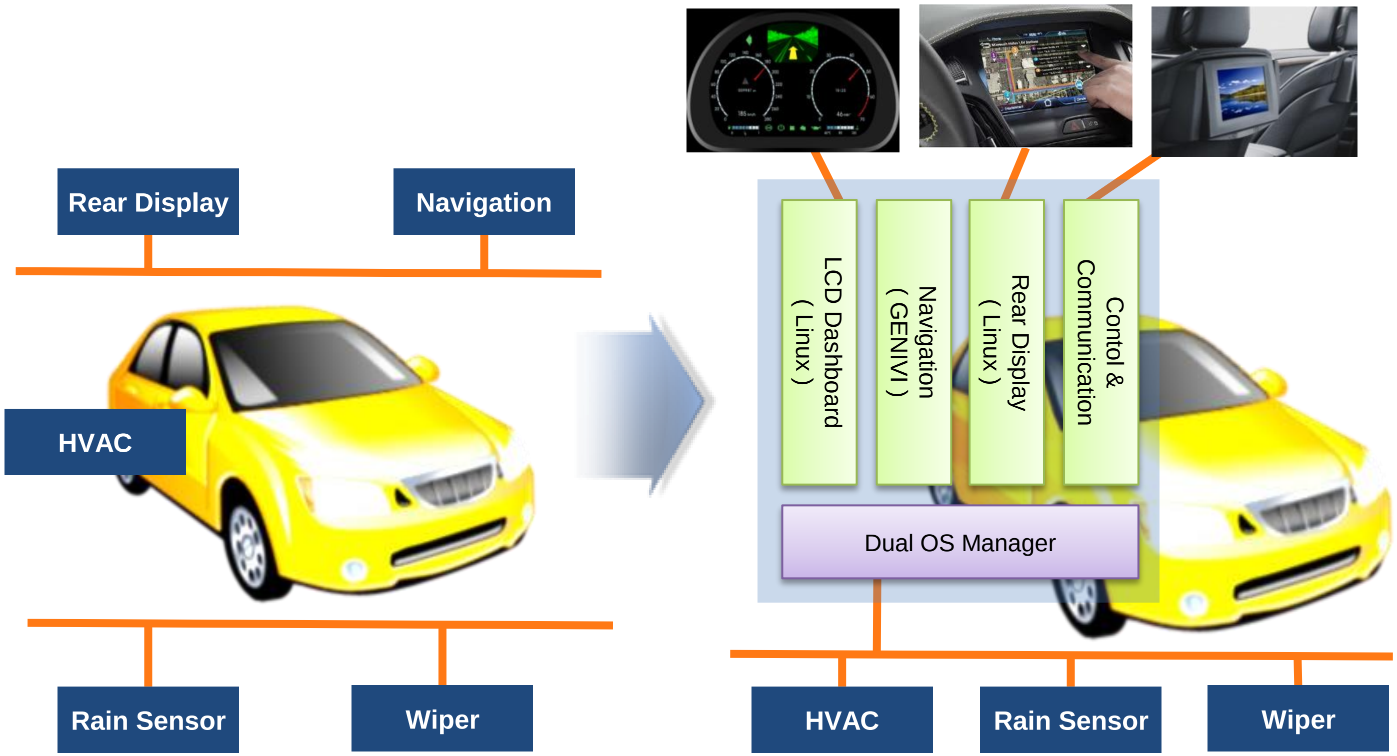
← Realtime Kernel Failure Detection

← Realtime Error Isolation and Healing

Highly Reliable Embedded System

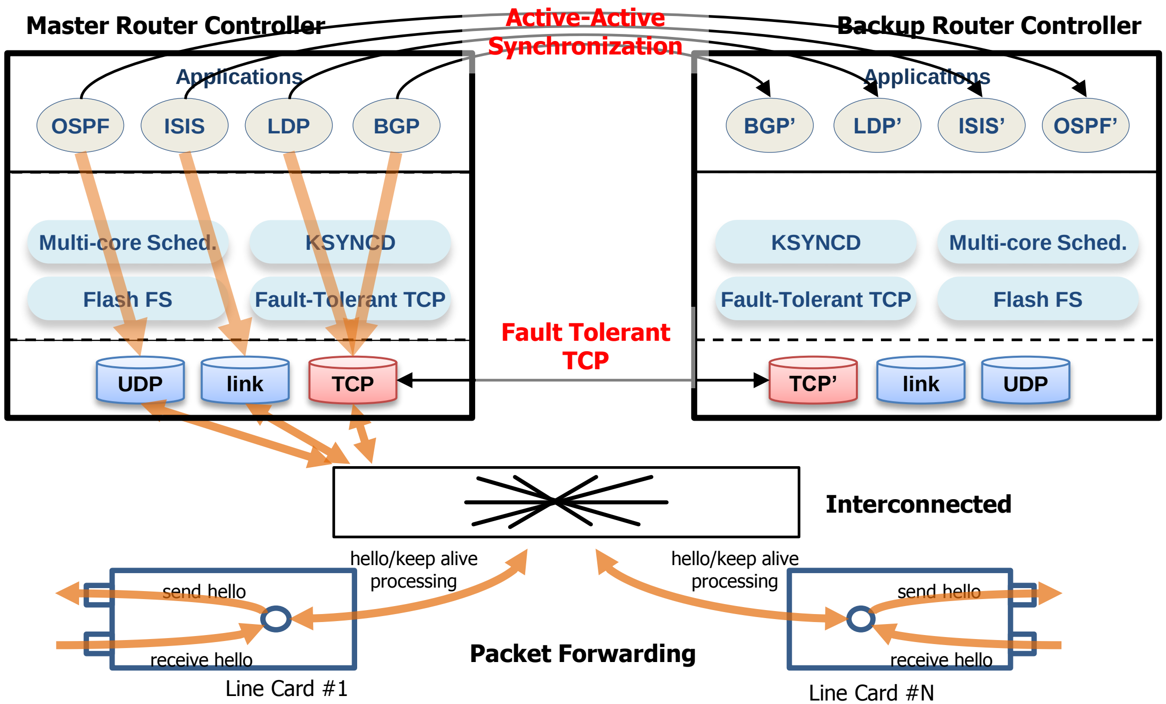
Application to IVI

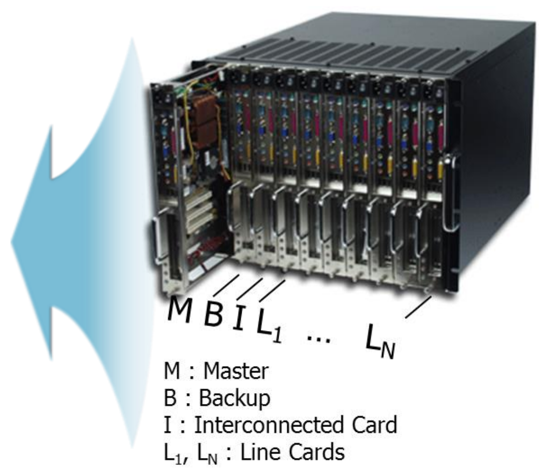
Make IVI to provide non-stop service experience even for a partial failure by fail-over



Architecture

Carrier Grade High available & High Performance Linux Technology Development

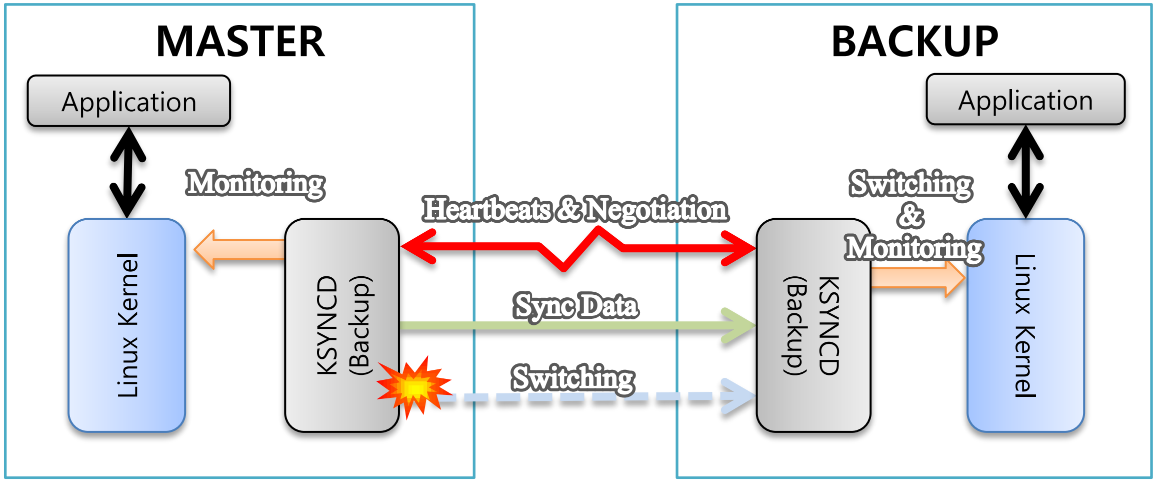




NSR: Non Stop active Router

KSYNCD

- Features
  - Heartbeat exchange interval is less then 1, while keeping system load low
  - Minimize mode switching
  - Control application's start/stop and sync'ing
  - Health monitoring & self healing
  - Kernel data sync'ing




THANK
YOU!