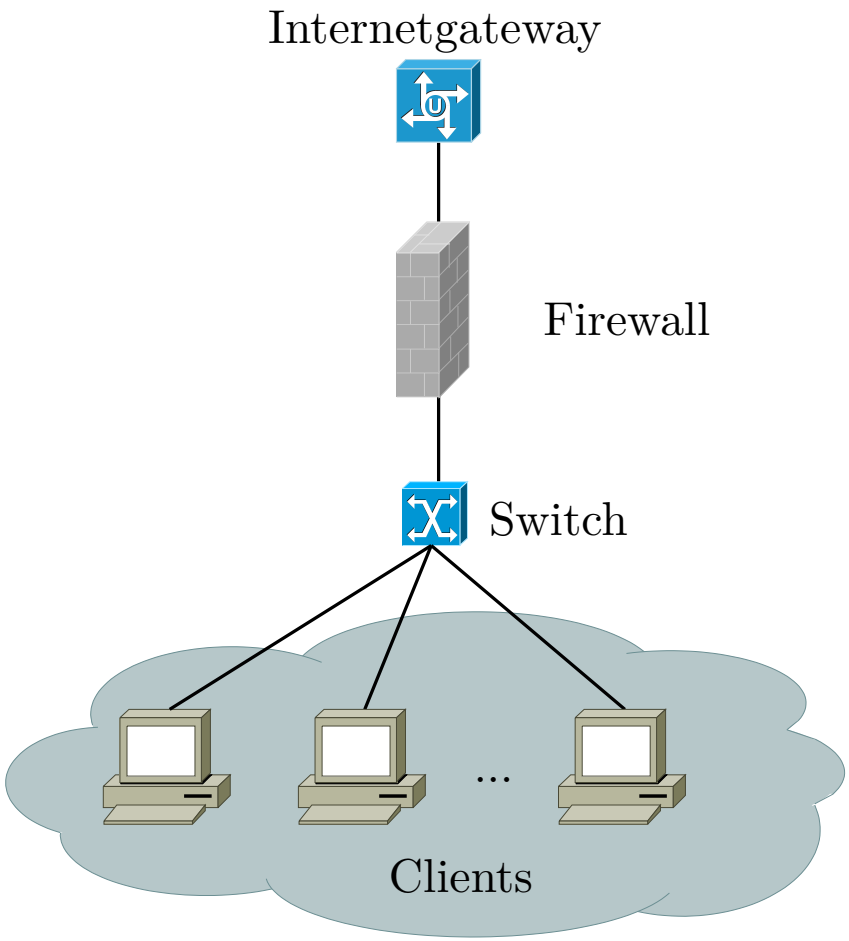
Stateful Firewalling on Active-Active Clusters

Ben Sartor

Department of Computer Science, Informatik 4
RWTH Aachen University, Ahornstr. 55, 52074 Aachen

Aachen, October 11, 2007

High Availability Firewall Cluster



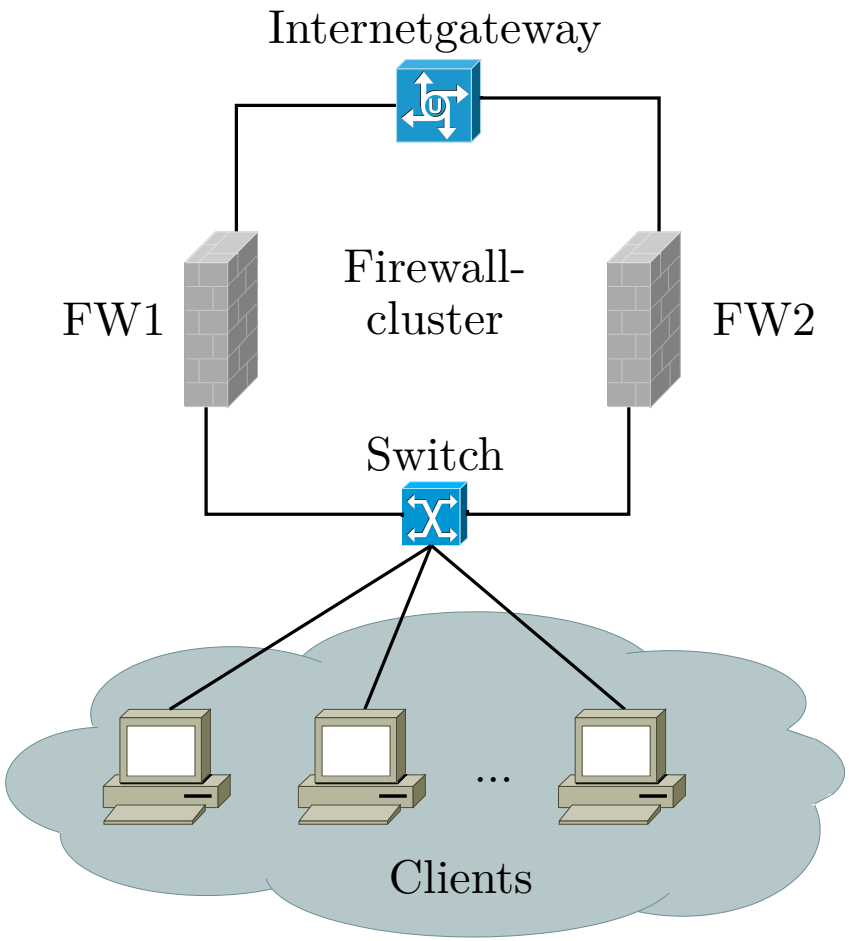
Single Point of Failure

- Central position in the network

Activ-Passive Cluster

- Many nodes, but only one active
- Failover of the master node
  - ⇒ Backup node becomes master
  - ⇒ Often connection loss

High Availability Firewall Cluster



Single Point of Failure

- Central position in the network

Activ-Passive Cluster

- Many nodes, but only one active
- Failover of the master node
  - ⇒ Backup node becomes master
  - ⇒ Often connection loss

Proposed Solution

Properties

- Active-Active Cluster
- Connections distributed evenly over the cluster
- Dynamically add/remove nodes

Advantages

- More efficient use of hardware resources
- On-the-fly integration of nodes
- Higher availability
  - ⇐ Not all connections affected at a node's failure
  - ⇐ No connection loss
  - ⇐ All nodes monitored

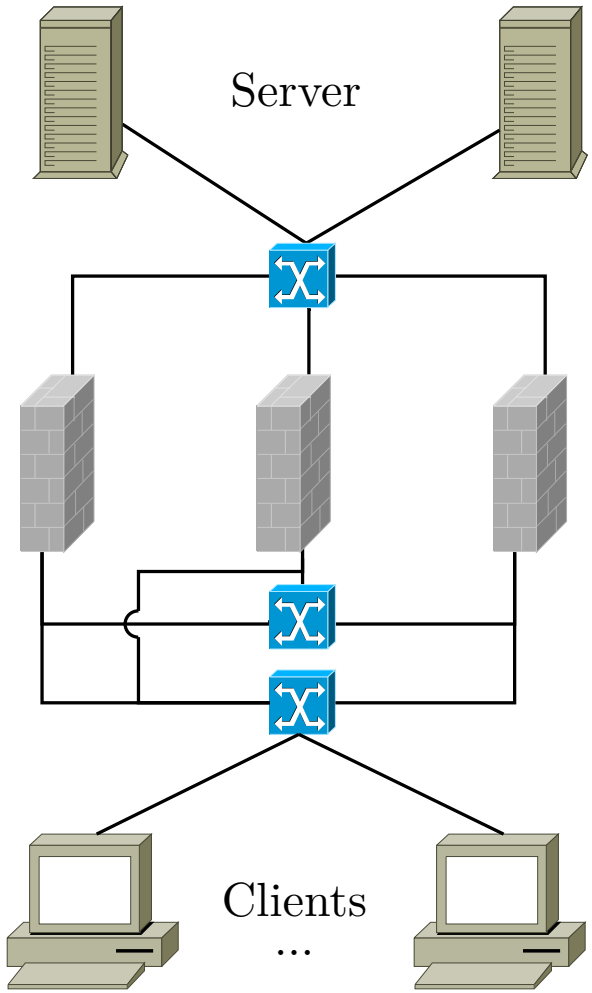
Demonstration in a virtual test environment

Kernel module `clusterdev`

- Based on `clusterip` from Harald Welte
- Virtual network interface

`availability-manager`

- Availability monitoring
- Connection destribution & connection synchronization



Basics

ARP

- ARP Requests
  - Ask for a MAC address to an IP address
  - Received by all computers in the LAN
- ARP Responses
  - MAC address in payload
- Gratuitous ARP
  - ARP Response with no ARP Requests
  - Take over IP addresses (e. g. at a failover)

Connection Tracking Table

- Enables stateful firewalling and NAT

Linux Firewall Cluster

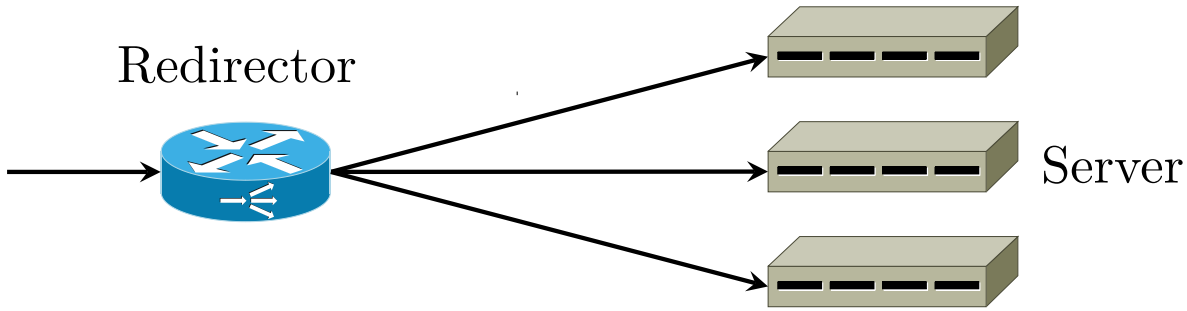
Active-Passive Cluster

- `keepalived` (VRRP), `heartbeat`, ...
- Monitoring availability of one node
- Failover: IP takeover with gratuitous ARP

Connection Tracking Table Synchronization

- `ct_sync`
  - 2002/2003 kernel module by Harald Welte & Krisztian Kovacz
- `libnetfilter_conntrack`
  - Editing of Connection Tracking Table from userspace
  - Userspace application: `conntrackd` (P. Neira, 2006)
  - Januar 2007: Pablo Neira Ayuso announces stable API

Linux Active-Active Clusters



- One IP address, many cluster nodes
- Until now Redirector
  - Distributes Connections among cluster nodes
  - Single Point of Failure
- Redirectorless Active-Active Cluster
  - ARP Balancing
  - MAC Multicasting

ARP Balancing vs. MAC Multicasting

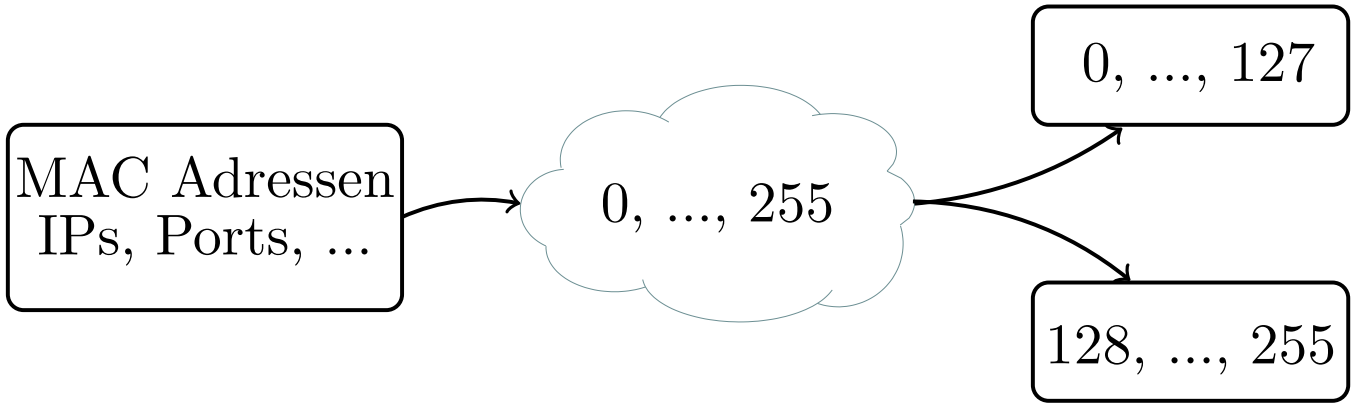
ARP Balancing

- ARP Response: MAC address of single node
- Packets of one client received by one node
- Each node decides about the handling of each client

MAC Multicasting

- ARP Response: Multicast MAC address
- All pakets received by all nodes
- Each node decides about the handling of each connection

Handling Decision



- Hash function: Connection data $\longrightarrow$ Responsible Range
- Connection data

  ARP Balancing: MAC address of sender

  MAC Multicasting: IPs, ports, ICMP-ID, . . .)
- Each node is assigned a subset of the whole Responsible Range
  - Based on this it decides about handling

ARP Balancing vs. MAC Multicasting

Failure of a node

- Remaining nodes take over the whole Responsible Range
- Particularity of ARP Balancing
  - Gratuitous ARP needed
    - 1 remaining node: Multicast
    - n remaining nodes: Unicast

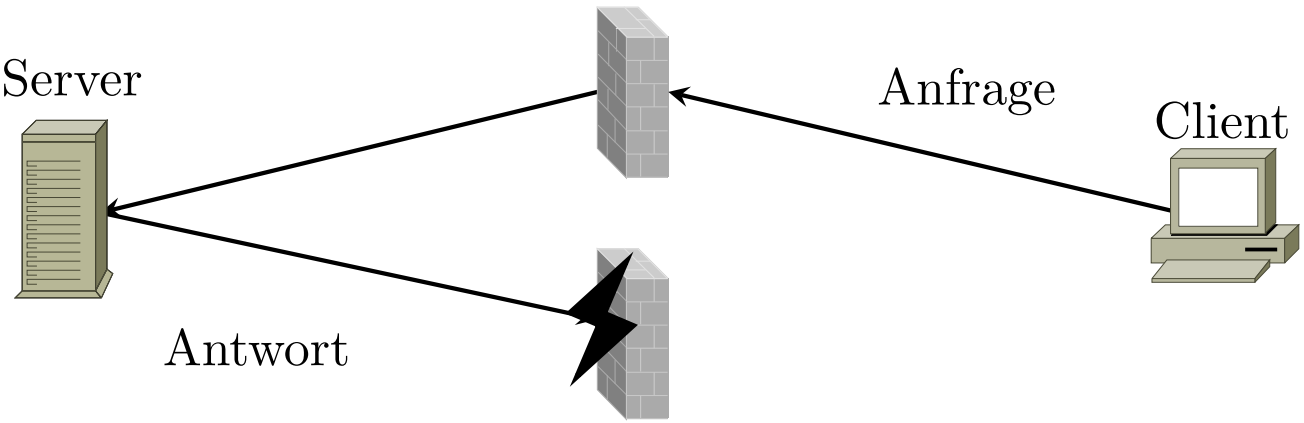
  $\Rightarrow$ Nodes need to know MAC addresses of all clients

ARP Balancing vs. MAC Multicasting

Connection distribution

- MAC Multicasting
  - Higher netload
  - $\Leftarrow$ All cluster nodes receive all packets
- ARP Balancing
  - Lesser netload
  - $\Leftarrow$ Clients are assigned to exactly one node
  - $\Leftarrow$ Use of ARP Balancing impossible, if on one side of cluster only one machine with only one MAC address exists

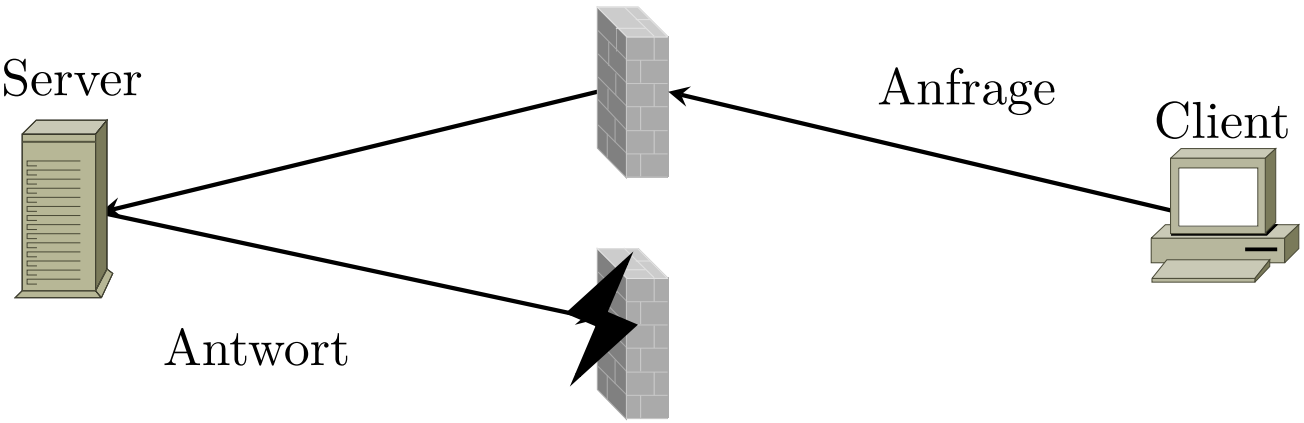
Problem with stateful firewalling



Problem because of the hash function

- Packets of one connection may be handled by different nodes
- Connection orientated packet filters may drop unknown packets

Problem with stateful firewalling



Solution 1: Synchronization

- Synchronization of Connection Tracking Table
- Each node knows about each connection
- Synchronization time critical

Solution 2: Connection-based

- Nodes take Connection Tracking information into account
- Only possible with MAC Multicasting

ARP Balancing vs. MAC Multicasting

Connection-based MAC Multicasting

- – High netload
- \+ Simple failover
- \+ Allows clusters with only one machine (1 MAC address) on one side of the cluster
- \+ Synchronization of Connection Tracking Table not time critical
- \+ Influence on acceptance probability with Responsible Range
- \+ Transfer of Connections possible by editing Connection Tracking Table

Kernel module `clusterdev`

Requirements

- Connection-based MAC Multicasting
  - TCP connections and forwarding with multicast MAC
  - Packet filter decisions and responses to ARP requests
  - Register multicast MAC addresses for receiving

  $\Rightarrow$ Kernel module

Implementation

- Multiple virtual network interfaces
  - Multicast MAC address, multiple IP addresses
  - Associated with real network interface
- Management of Responsible Range with the `/proc` filesystem

availability-manager

Requirements

- Dynamically add & remove nodes
  - Availability monitoring of all nodes
  - Distribution of Responsible Ranges
  - Synchronization of Connection Tracking Table
- Extensibility
  - Pluggable: Synchronization of other servers

$\Rightarrow$ Layer model

Synchronization Layer	Clusterdev Protocol: Distribution of Responsible Ranges Conntrack Protocol: Synchronization of Connection Tracking Table
Reliable Broadcast Layer	Own Status up/down Node up/down RBSend/RBRecv Out-of-Sync
Availability Layer	Own Status up/down Node up/down URSend/URRecv

Availability Monitoring

Monitor other nodes

- All approaches for availability monitoring are based on a timeout mechanism (Tanenbaum)
- Each node sends heartbeat packets (IPv4 broadcast)
- NodeID: IP address in the firewall network

Monitor own status

- Internal status of the network card
- Status of the network interface

Reliable Broadcast Protocol

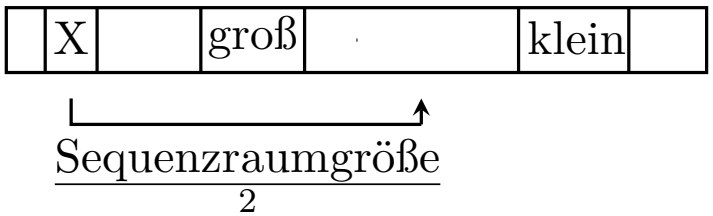
Requirements

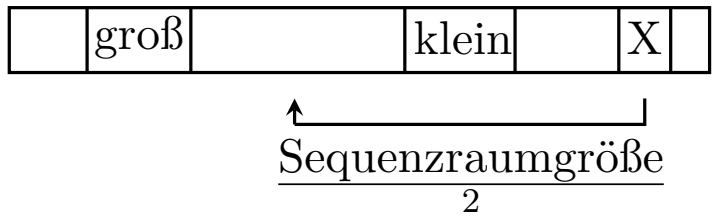
- Low packetloss in LANs
  - $\Rightarrow$ Lightweight protocol
- Best effort protocol
  - Messages may get lost, if the sender dies
  - $\Rightarrow$ Each node saves its sent packets
- FiFo Order
  - All packets of one sender are received in the order sent
  - With many senders and different receivers no guarantee about global order
- NAK based protocol
  - NAK based protocols scale better than ACK based ones (Tanenbaum)

Reliable Broadcast Algorithm

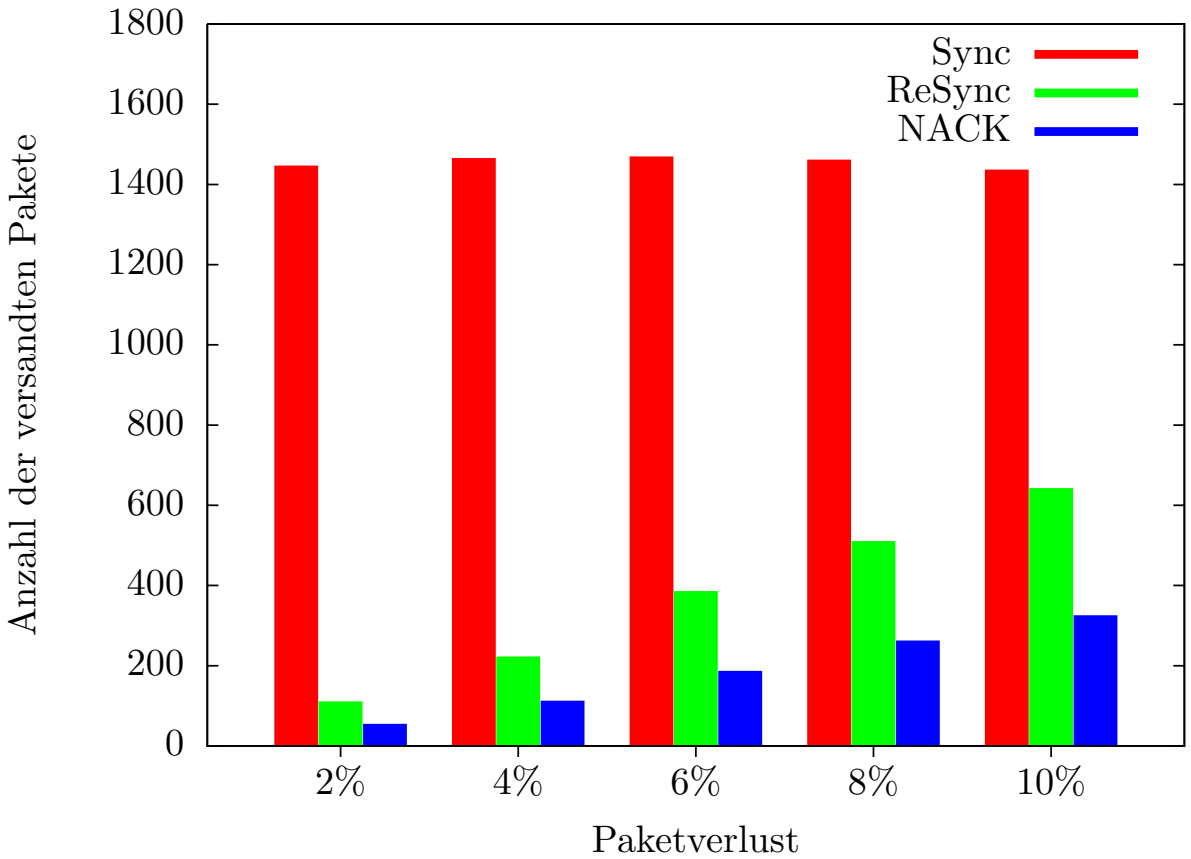
Implementation

- Out of Sync
  - Number of lost packets $>$ Sender buffer size
- Synchronization at beginning
  - A new node is not signaled to the next layer until it has sent a packet with the correct cluster size
- Sequence numbers





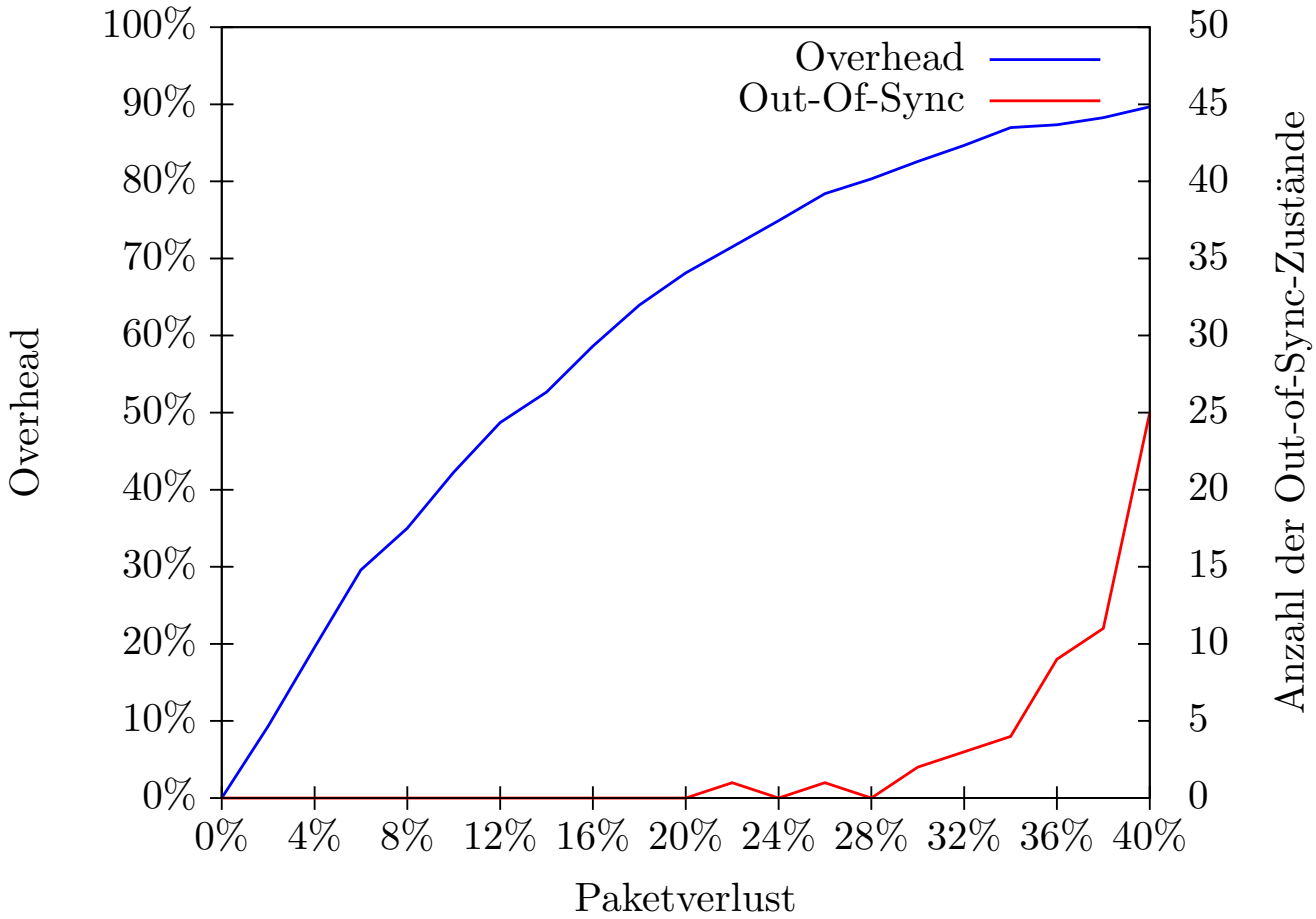
Measurement with induced packet loss



Test environment

- Test plugin
  - All 100 ms
- 3 prozesses
- Measurement
  - In each case 1 min

Measurement with induced packet loss



$$\text{Overhead} = \frac{\text{NAK+ReSync}}{\text{NAK+ReSync+Sync}}$$

Clusterdev Protocol

Requirements

- Distribution of the Responsible Range
  - Failure of a node: Remaining nodes share its range
  - New node: Assign a subset of the Responsible Range
- Targeted distribution (Cluster Configuration)
  - *complete:* Complete Responsible Range is assigned to at least one node
  - *pairwise disjoint:* No intersection of node ranges
  - *equally distributed:* Ranges assigned to nodes almost same size

Clusterdev Protocol: Requirements analysis

Completeness

- Cluster configuration always *complete*
- Otherwise connections may be unhandled

Change of cluster configuration

- Not possible on all nodes at the same time

$\Rightarrow$ A node only releases part of its Responsible Range if another node claims to have it

$\Rightarrow$ Double assigned Responsible Ranges temporarily allowed

Clusterdev Protocol: Requirements analysis

	C_1^3	C_2^3	C_3^3
K_1	$0 \ldots 84$	$0 \ldots 84$	$0 \ldots 84$
K_2	$85 \ldots 169$	$85 \ldots 169$	$85 \ldots 255$
K_3	$170 \ldots 255$	$170 \ldots 255$	$170 \ldots 255$

	C_1^3	C_3^3
K_1	$0 \ldots 127$	$0 \ldots 169$
K_3	$128 \ldots 255$	$170 \ldots 255$

Cluster Configuration

- Check completeness & pairwise disjoint
  - $\Rightarrow$ Each node knows the ranges of the other nodes
- Failover
  - This information suitable to only a limited extent

Clusterdev-Visual-Tester

Example

Clusterdev Protocol: Implementation

	C_1^3	C_2^3	C_3^3
K_1	$0 \ldots 84$	$0 \ldots 84$	$0 \ldots 84$
K_2	$85 \ldots 169$	$85 \ldots 169$	$85 \ldots 169$
K_3	$170 \ldots 255$	$170 \ldots 255$	$170 \ldots 255$

Enhancement of Requirements

- The aimed cluster configuration is also *sorted*
  - Responsible Ranges ordered by NodeID

$\Rightarrow$ Aimed cluster configuration: unique

$\Rightarrow$ Each node can be assigned a unique aimed Responsible Range

  - Clustersize
  - Position in the cluster (NodeID)

Clusterdev Protocol: Implementation

Clusterdev Algorithm

- Loop
  - myRange = myRange ∪ myAimedRange
  - SendBroadcast myRange
- ReceiveBroadcast(NodeRange)
  - Save NodeRange for sender node
  - T = myRange ∩ NodeRange
  - IF T NotIn myAimedRange THEN Delete T from myRange
  - Check completeness
- Failover(NodeID)
  - myRange = myRange ∪ myAimedRange

Clusterdev Visual Tester

Clusterdev Protocol

Clusterdev Protocol: Analysis

Distribution of Responsible Ranges

- Each node has received at least one broadcast message from all other nodes
- Each node has assigned his aimed Responsible Range
  - Minimum
    - Each node has send a broadcast message
    - Each node has added his aimed range to his Responsible Range
  - Maximum
    - Release of subranges, that are not in the aimed range of a node, when receiving a broadcast message
    - Having received a message from each node

    $\Rightarrow$ Minimum aimed range of each node

    $\Rightarrow$ With own aimed range *complete*

Conntrack Protocol: possible Errors

Multiple handled connections

- Responsible Ranges may be assigned to multiple nodes

$\Rightarrow$ Connections may be handled by multiple nodes

$\Rightarrow$ All but one node have to release this connection

Avoiding packetloss at a failover

- Failure of a node
  - The remaining nodes share its connections
  - Adding corresponding entries to the Connection Tracking Table

$\Rightarrow$ Node needs to know the relevant Connection Tracking Entries of the other nodes

Conntrack Protocol: Synchronised Data

Relevant Connection Tracking Entries

- Contain cluster IP addresses
- Protocol configurable: TCP, UDP, ICMP, ...
- `EXPECTED` entries not synchronised

Attributes of an entry

- Relevant
  - Connection characteristic data: IPs, Ports, ...
  - Connection status
- Not relevant
  - Packet counter, Timeout, ...
  - Changes not synchronized

Conntrack Protocol: Implementation

Connection Tracking Synchronization

- Message type
  - NEW: Connection characteristic data + connection status
  - NEW_RESYNC: NEW message with Resync-Flag
  - DELETE: Checksum(connection characteristic data)
  - UPDATE: Checksum + connection status
- Send all relevant Connection Tracking Entries
  - If Reliable Broadcast Layer signals Out-of-Sync
  - If a new node appears
  - If the periodic consistence check fails
  - NEW_RESYNC, NEW, NEW, ...

Analysis

Tests

- Component tests
  - Implementation tested with extra applications
    - Clusterdev Protocol, Reliable Broadcast Protocol
  - Availability monitoring tested with multiple computers
  - Kernel module `clusterdev`
    - Different hardware: PCs, switches, QSC Router, ...
    - Functions: Responsible Ranges, IP address (`ping`)
- Overall test
  - Failover test with connection takeover
    - HTTP, FTP, SSH, SSL
  - Practical test
    - Stability problems with high load

  f

Discussion

Active-active Problem

- If a cluster runs at full capacity the failure of a node can lead to packetloss
  - $\Leftarrow$ Load without the fallen out to high

Clusterdev Protocol

- *Evenly distributed* Cluster configuration

$\Rightarrow$ Similar number of connections on each node

$\not\Rightarrow$ Similar load on each node

- Not using any guarantee of Reliable Broadcast Layer

$\Rightarrow$ Implementation in Distribution Layer

Discussion

Separate synchronisation net

- Heartbeat packets only in synchronisation net
- Otherwise connection – and range takeover possible
- Switch: Single Point of Failure
  - Wireless: WLAN, Bluetooth

Encryption Layer

- Loss of synchronisation layer
- Monitor other networks

Future

IPv6 support

- Enhancement of kernel module `clusterdev`
  - Reusing Responsible Ranges
- Enhancement of `availability-manager`
  - Enhance Conntrack Plugin
  - Encryption already included

More Protocols

- E. g. Master Election Protocol
  - MAC-Multicasting vs. ARP IP Sharing (keepalived)
  - Enhancement of Clusterdev Protocol vs. separate protocol
  - Aktiv-passive services
    - Cluster state information, DHCP server, ...

Synchronisations-schicht	Additonal Protocol ... Conntrack Protocol: Synchronization of Connection Tracking Table
Distribution Layer	Clusterdev Protocol: Distribution of Responsible Ranges; Reliable Broadcast: Own Status up/down Node up/down RBSend/RBRecv Out-of-Sync
Availability Layer	Own Status up/down Node Up/down URSend/URRecv
Encryption Layer	URSend/URRecv

Summary

Connection-based MAC Multicasting

- Different nodes handle different connections

$\Rightarrow$ One machine possible on one side of the cluster

Active Active Firewall Cluster

+ On-the-fly enlargement
+ Higher availability
  - Availability monitoring all nodes
  - No connection loss at a failover