Implementation Aspects of Communication Protocols in Active Network Nodes for the Dynamic Installation of Performance Enhancing Protocols[1]

Michael Eyrich, Morten Schläger, Prof. Dr. Adam Wolisz

Telecommunication Networks Group, Technische Universität Berlin
eyrich@ee.tu-berlin.de

Feb. 13 – Feb. 15, 2002

[1]Work in progress of the FlexiNet Project;
Acknowledgments: Bundesministerium für Bildung und Forschung (BMB+F)

Outline

- Why Active Networks?
- ReSoA Architecture
- Implementation Aspects
- Measurement Evaluation
- Outlook

What's it all about?

Performance Enhancing Proxies are an appealing solution for:

- Coupling of heterogenous networks (e.g. wireless Internet access).
- Many different (usefull) proxies developed $\Rightarrow$ not yet deployed?

Active Networks as a deployment booster . . .

- Scanning of network traffic (network attacks, e.g., DoS)
- Flow conversion (Transcoding)
- Proxies
  - Can it be integrated in an AN?
  - Which problems are raised by the integration?
  - How does it perform?

A case study within FlexiNet at TU Berlin

- Integration of ReSoA into AN environment

Our Performance Enhancing Proxy

ReSoA: REmote SOcket Architecture

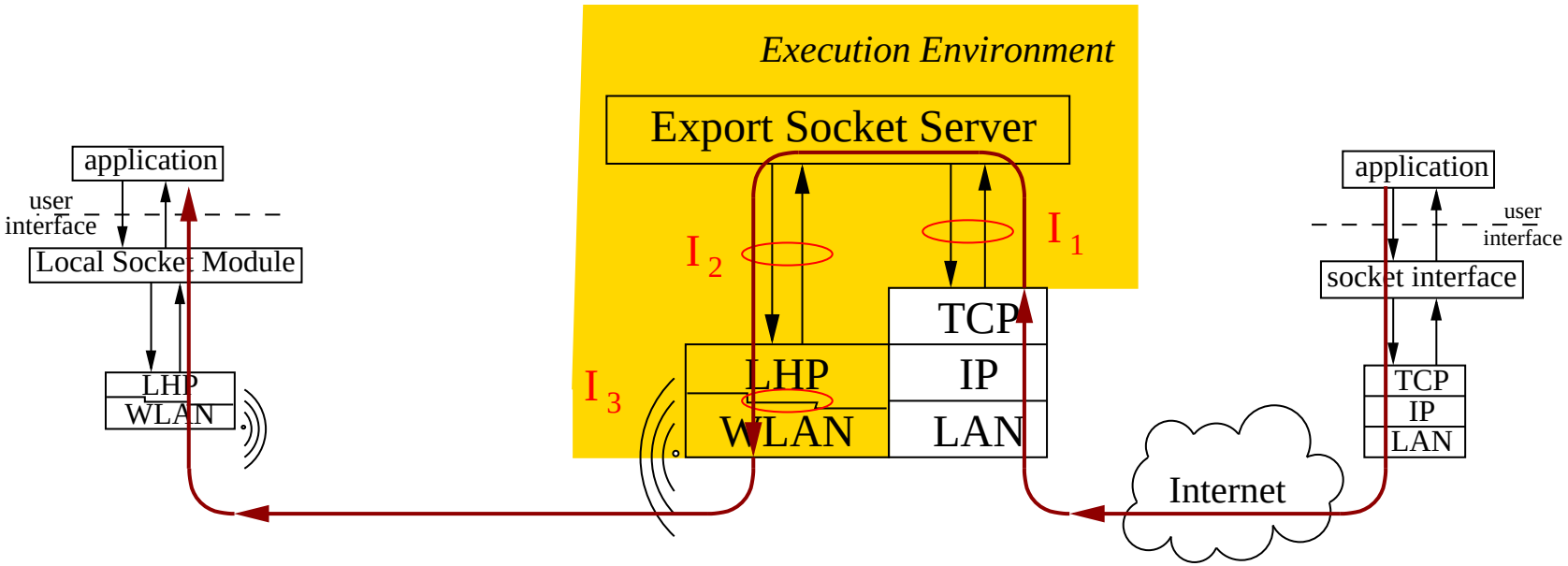
- Transport Protocol Proxy
  - Improved performance of wireless Internet access
  - Export of socket interface
- Flow decoupling:
  - LHP between ReSoA server and wireless station (end-system)
  - TCP between correspondent host and ReSoA server
- Last Hop Protocol (LHP)
  - Reliable service in case of TCP, flow dependent in case of UDP
  - Technologie dependent
  - Exchangeable
    - can be provided by manufacturer

ReSoA– Model

wireless system

access point = active node

correspondent node



Necessities for proxy downloads

Execution Level

- Kernel vs. User space

Interfaces to the system

- Packet multiplexing and demultiplexing
- Access to service of static protocol stack (e.g. transport layer)
- Interface to control behavior of static protocol stack (e.g. delay ACKS)
- System services (e.g., timer, process interaction)

Necessities for proxy downloads (cont.)

Signalling environment

- Where to find a proxy
- How to communicate version, architecture, etc.

Security and Fairness

- Is it safe to execute the code I got?
- Enforcement of beeing cooperative (memory, processing time, . . .)

Execution Environments

Feature	Linux Kernel	AMnet	Crossbow Cobra	required for PEP
Signalling	–	✓	–/??	(✓)
Security/Fairness	–	✓	–	(✓)
Modularized	✓	✓	✓	✓
Module instances	–	–	✓	–
De-/Multiplexing	✓	✓	✓	–
Transport layer data interface	(socket)	(socket)	(socket)	✓
Transport layer control interface	–	–	–	(✓)
Network layer interface	✓	✓	✓	✓
MAC layer interface	✓	✓	✓	✓

→ Specialized EE only for management (signalling, security, fairness)

→ Additional services needed not provided by EEs.

Implementation Design

Kernel modules

- ReSoA server and client
- Last Hop Protocol (LHP) environment
  - Registration of and wrapper to LHP
  - Protocol and address independent interface to LHP
- Last Hop Protocols
  - currently IPLHP, i.e., based on IP
  - LLP: MAC Link Layer Protocol extension

Interfaces

- Transport layer access: sockets (I_1)
- Internal Interface: ReSoA– LHP (I_2)
- De-/Multiplexing: New protocol familiy for LHP (I_3)

Interface to local TCP

What does it mean to take Sockets?

- Socket requires copying of data (protection of kernel space)
  - no way to inject skb's into TCP stack
- Socket requires process context (I_1)
  - Queueing of packets necessary (Socket callback lock)
  - Explicit tagging of process to get runable
  - Explicit scheduling of process
    (non-preemptive scheduling for threads)

  $\longrightarrow$ Delay therefore not predictive (load dependent)
- Socket structure depends on configured protocols!
  (Bug or Feature?)
- Socket allows for waiting on events (reception of data, ...)

In-kernel socket Interface

Access to functions

- Standard socket creation (`sock_create()`)
- Direct function calls (not via `sys_socketcall`)

Access to data

- sock_sendmsg requires an `iovec`
- Explicit switch to kernel data segment required

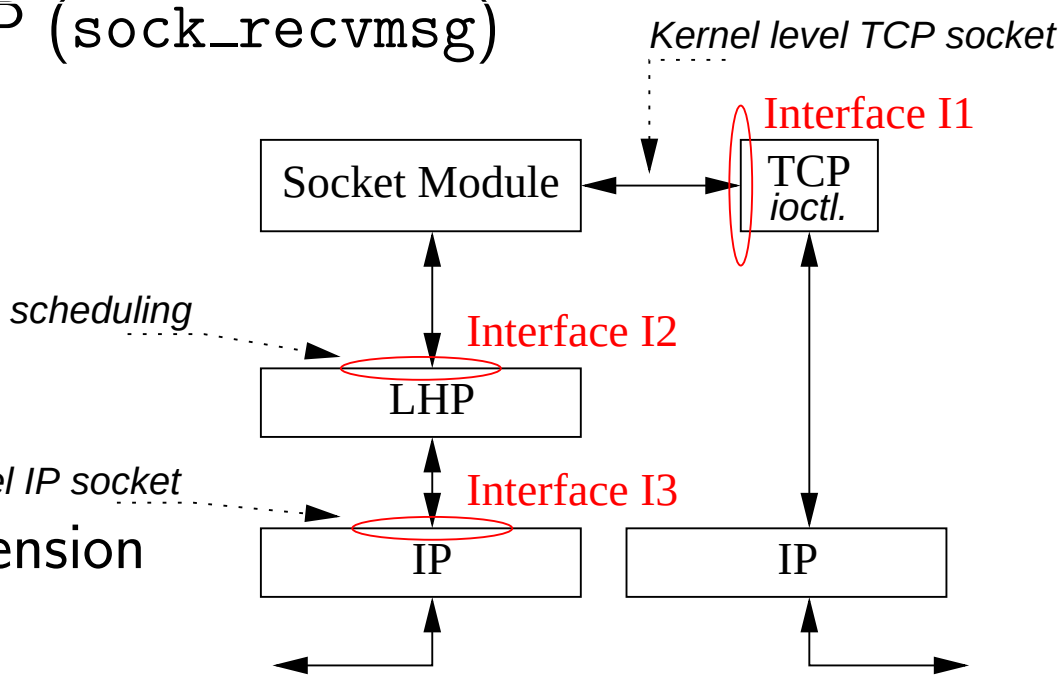
Example

```
mm_segment_t oldfs;
struct msghdr msg;
...
oldfs = get_fs(); set_fs(KERNEL_DS);
err = sock_sendmsg(sock, &msg, len);
set_fs(oldfs);
```

Measurement Environment

What is to be measured?

- Data transfer from FH to WH via AN
- I_1: Time to read data from TCP (`sock_recvmsg`)
- I_2: Scheduling delay
- I_3: Total time within ReSoA (`sock_sendmsg`)

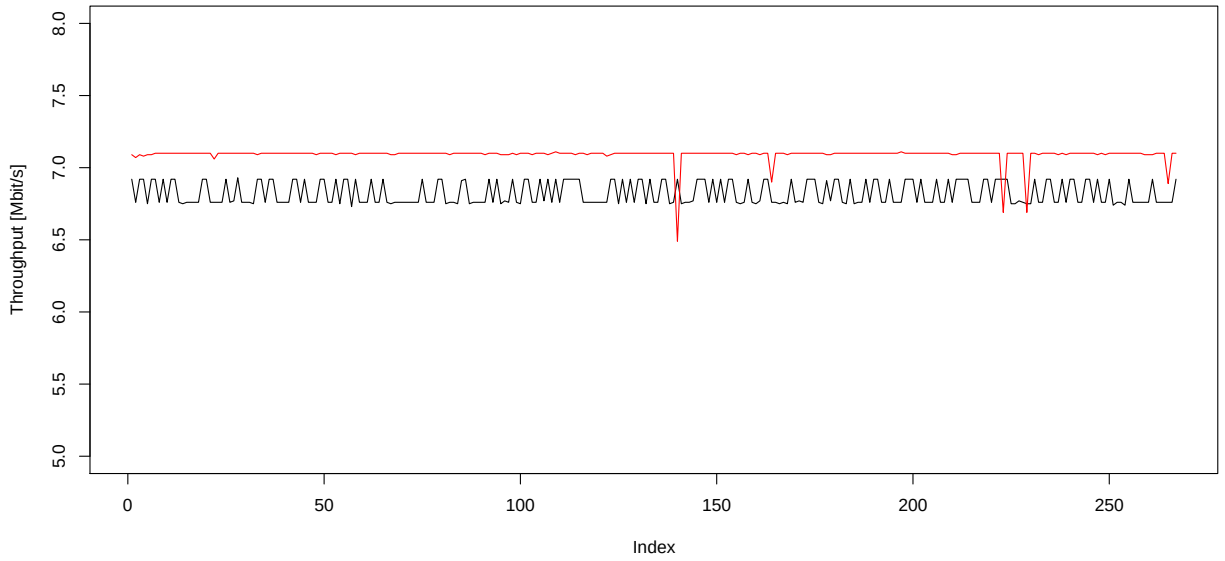


Snuffle

- a protocol state monitoring extension
- In-kernel tracepoints
- In-kernel buffer to keep data
- Java application for data collection (read at regular intervals)
- How to measure time between tracepoints? $\longrightarrow$ `skb` address

General throughput

Comparision TCP vs. ReSoA

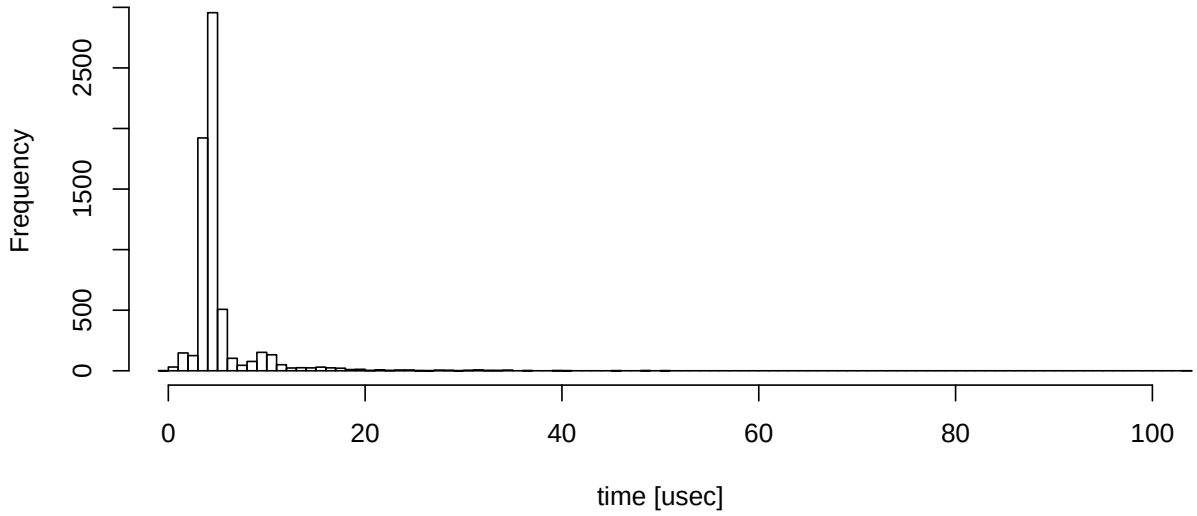


Comparision

- Implementation of ReSoA (red) behaves well with regard to throughput of TCP (black)

Socket receive (I_1)

Histogramm

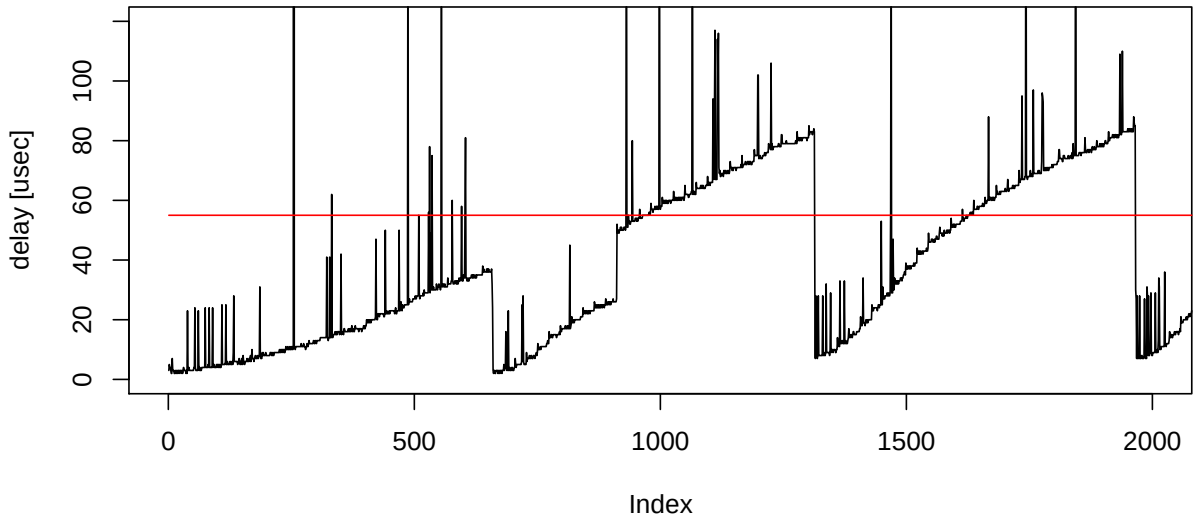


Time to copy from socket to skb

- Median 5 μsec for average size of 1448 kbyte

Scheduling delays (I_2)

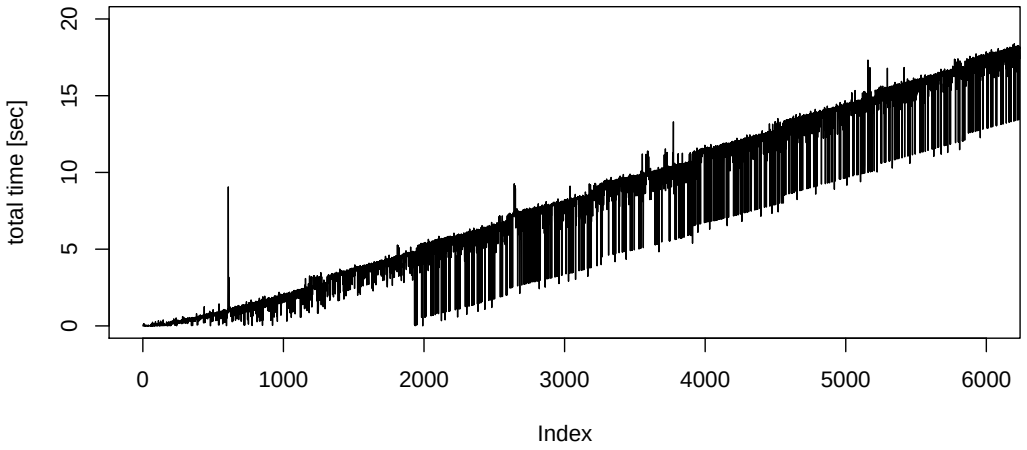
Athlon/1500



Scheduling delay (μsec) per packet

- Median: 55 μsec (red line)
- saw-tooth shape caused by increasing load

Stay-time within ReSoA framework



Per Packet stay-time within ReSoA

- Links of different quality (100 MBit/s vs. 10 MBit/s “wireless”)
- backpressure too slowly propagated to TCP (full socket receive buffer)?
- We need an interface to control the sending speed of TCP ACKs
- Work in progress!

Conclusion

We have . . .

- a dynamic loadable implementation of ReSoA
- an internal interface for the exchange of a LHP (address and protocol independent)
- a measurement setup including in-kernel tracepoints

We conclude . . .

- Integration is possible
- Throughput behaves well
- System does not scale because of copying and scheduling
- Neither environment provides TL interface with copy-avoidance
- Kernel patches required to modify TCP's behavior (netfilter?)
- All environments provide network layer access

Outlook

Further work

- LLP integration
- Socket extensions for reduction of copy operations
- Signalling integration
- Parameters for signalling (e.g. Kernel version)
- Name space collisions between modules

Fairness?

- Avoid starving of other processes
- Cleanup memory
- Never sleep in interupt context

Modifications to protocol state

- filter rules for delaying ACK