Design and Implementation of a Python-Based Active Network Platform for Network Management and Control

Florian Baumgartner

Institute of Computer Science and Applied Mathematics
University of Bern

Department of Computer Sciences
Purdue University

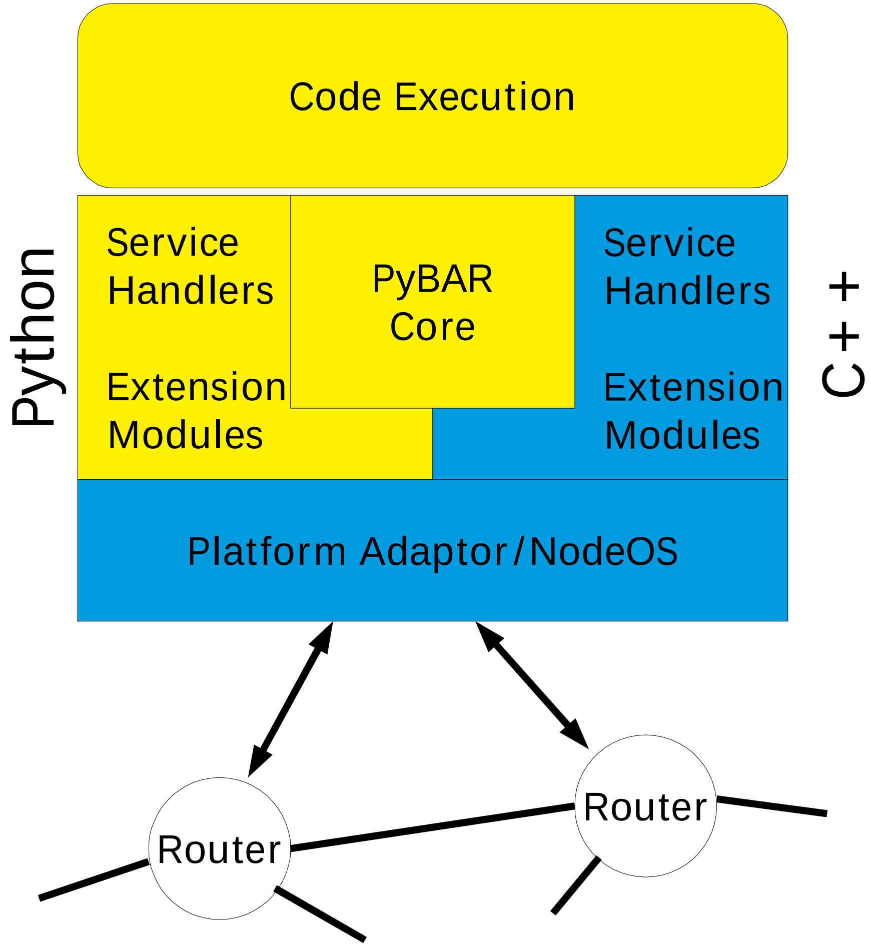
Motivation

- Development of a platform especially for the purpose of network configuration.
- The system's focus is not the development of huge distributed systems, but a lightweight, easy to use framework to adjust TC systems or to collect information within the network.
- To provide as much flexibility and modularity as possible.
- Integration of existing applications/libraries.

Why Python ?

- Properties like most modern interpreted languages
  - portable bytecode, OO (not only), restricted execution environments
- Advantages of Python:
  - Prototyping language supporting high level data types -> rapid prototyping. (glue language)
  - Python is very extensibility
    - seamless and flexible integration of native code modules.
    - even modifications of Python internals are possible.
  - Python programs are three to five times smaller than in Java.

PyBAR Architecture

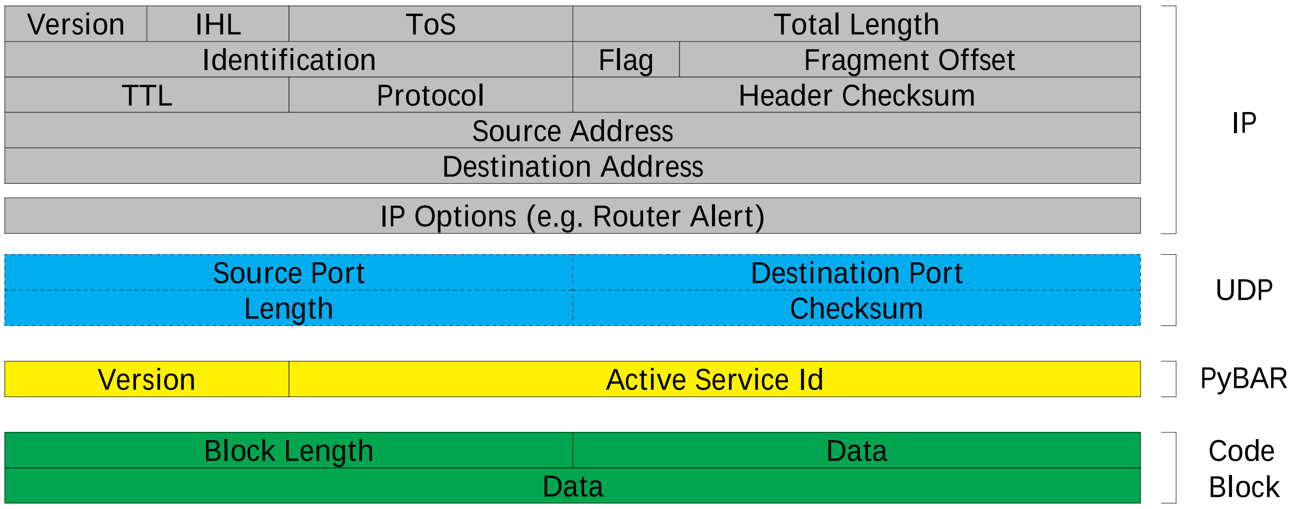


- PyBAR is more a framework, which can be adapted by integrating modules from a module pool. (e.g. encryption, resource control)
- Packets may contain code or can be directly processed by a service handler.
- Native modules allow a complete "Python-free" processing of packets.
- Thin NodeOS uses various kernel interfaces (tc, filtering). Modules provide high level functionalities.
- (One PyBAR can control multiple routers.)

Addressing & Packet Transport

- Direct UDP/IP
  - Addressing of a specific device
- Router Alert
  - processing overhead in conventional routers
- DSCP to trigger packet execution
  - can be used for direct addressing or for processing along a certain path.
  - DSCP can also be used to avoid loss of active packets in not-active routers
  - no processing overhead in not-active routers
- generic packet filter

PyBAR Packet format



Version | IHL | ToS | Total Length
Identification | Flag | Fragment Offset
TTL | Protocol | Header Checksum
Source Address
Destination Address
IP Options (e.g. Router Alert)
IP

Source Port | Destination Port
Length | Checksum
UDP

Version | Active Service Id
PyBAR

Block Length | Data
Data
Code Block

- PyBAR does not rely on a specific packet type (future system might use ANEP).
- Packet processing is left to the core. The current, very simple packet type is used to cause as less overhead as possible.

Security

- limited user group (administrators, daemons)
- Security modules to provides authorization/encryption mechanisms.
- Current security module is based on the RSA reference implementation and provides a high level interface for applications.
- Modular approach allows to realize different security concepts.
- Packets are processed in restricted execution environments.
- resource control by monitoring execution.

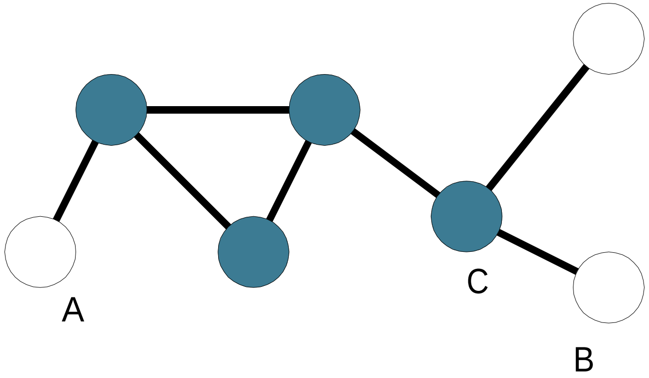
Differentiated Service Support

- no built in DS support (e.g. by the NodeOS)
- Differentiated Service support by extension module.
  - support for heterogeneous platforms (UniBe DS, VR) and networks
  - can be easily replaced
  - can provide a high level API instead of defining only fundamental commands.

init(<type>)	sets up the complete traffic conditioning components requires for DiffServ. with an appropriate scheduler, EF and AF queues, token bucket filters
setClassShares()	configures the bandwidth shares for the dfferent traffic types
mark(<>)	configures the Differentiated Services marker to mark specific flows with DSCPs

Application
Tunnel Endpoint Discovery

- Problem:
  - Tunnel set up process is sender driven, a matching end point is required.
  - If the receiver is not capable to handle the tunnel, an upstream node should be used.
- Solution:
  - Inject active packet with search pattern (decryption mechanisms).



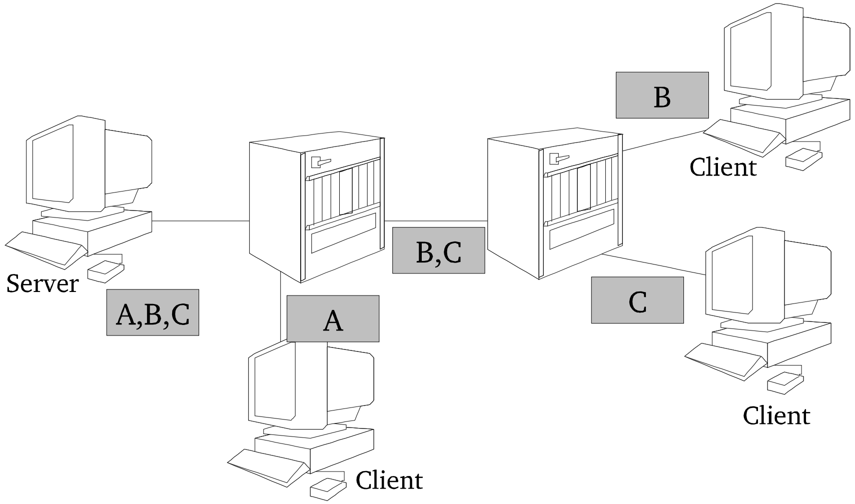
Application
Tunnel Endpoint Discovery

- Requests property list from router.
- If property list contains requested capabilities, a feedback packet is sent back to the tunnel start point.
- Tunnel start point may choose among the most appropriate end point.

```
class DiscoverEP(ARpacket):
    def __init__(self.packet):
        #get a list of router properties/services
        c=pad.getCaps()
        #if IPIP available, extract information from
        #code block and send feedback packet
        if c.count("IPIP"):
            src_info=unpack.loads(acpkt.cb(1))
            #generate and send feedback packet
            p=pad.UDPPacket()
            p.source=pad.hostip
            p.dest=src_info['tunnel_start']
            p.destport=src_info['portnumber']
            p.payload=pack.dumps('service':'IPIP',
                'tunnel_end':pad_host_ip,'time':pad.time)
            p.send()
        #forward original active packet
        acpkt.send()
        return
```

A Short Glance on Performance
A Simple Active Multicast Service

- Classical active multicast example.
- Send packet with multiple addresses.
- Packet is processed by service handler within the PyBAR.
  - pure Python SH
  - Python free SH



A Short Glance on Performance
Packet Rates

- UDP based, configurable video sender as traffic source.
- C++ version causes very limited overhead.
- Measurements with C++ limited by 100Mbps inbound /outbound link

Addresses	ms/packet	rate (inbound)
Python Module		
4	1	1000
8	1.7	580
16	2.2	454
C++ module		
4	0.01	>10000
8	0.03	>10000
16	0.05	>10000

Summary & Conclusion

- Python is less application and more prototyping oriented than Java. Support for rapid development of applications.
- It can provide modularity and allows to transparently integrate native code.
- The modular approach of the PyBAR allows to quickly integrate new concepts (e.g. for security) and to build specialized systems.
- Performance: Python-free processing path provides reasonable performance.