Programmable networks – what's the big deal?

Gísli Hjálmtýsson
Prof. of Computer Science
Networking Systems and Service Laboratory
Reykjavík University, Reykjavik

Overview

- Motivation, drivers and background
- The Pronto router
  - Programmable services
- The new principle of timescales
- Evolvable router architectures
  - Do network processors matter?
- Reflection - what have we achieved?
- Where do we go from here?
- Summary

Motivation

- Service deployment – rapid service creation
  - High margin for new services – competitive edge
  - Same drivers as for AIN in telephony
- Standardization takes too long
  - IETF's complexity is growing exponentially
  - So is time to standardize
  - Multicast, mobile IP, IPv6, etc. has taken forever
- Technology trends
  - Rapid growth in bandwidth and # of transistors (cycles)
  - Need for adaptation – the only certainty is change
  - Commoditization – of network hardware (routers)

Motivation

- Universal boarder gateway
  - Currently have a range of ad-hoc gateways (NAT, Proxy)
- Evolvable routers – adapt to changing hardware
  - Exploit capabilities of increasing flora of hardware
  - Revisit architecture of IP nodes
- Application driven evolution
  - Multicast and content distribution networks
- Reinvent the Internet model
  - Pipes and cycles
  - Traffic engineering is out
  - PlanetLab the mother of all overlays

Active and Programmable Networks

- We do program traditional networks
  - through standardization (IETF, ATM Forum)
  - or by dominant vendors (Cisco)
- The critical difference is …
- … who gets to program them
- … how they are programmed
- … how much consensus is needed
- … when and how fast they get programmed

Issues in network programmability

- Time scale of programmability
  - How fast, how often?
- Time scale of activity
  - Management, call holding, packet
- Model of programmability
  - Openness vs. Programmability vs. Activity
  - Control plane, Pronto, Capsules
- Interfaces
  - What is there? Are they opaque?

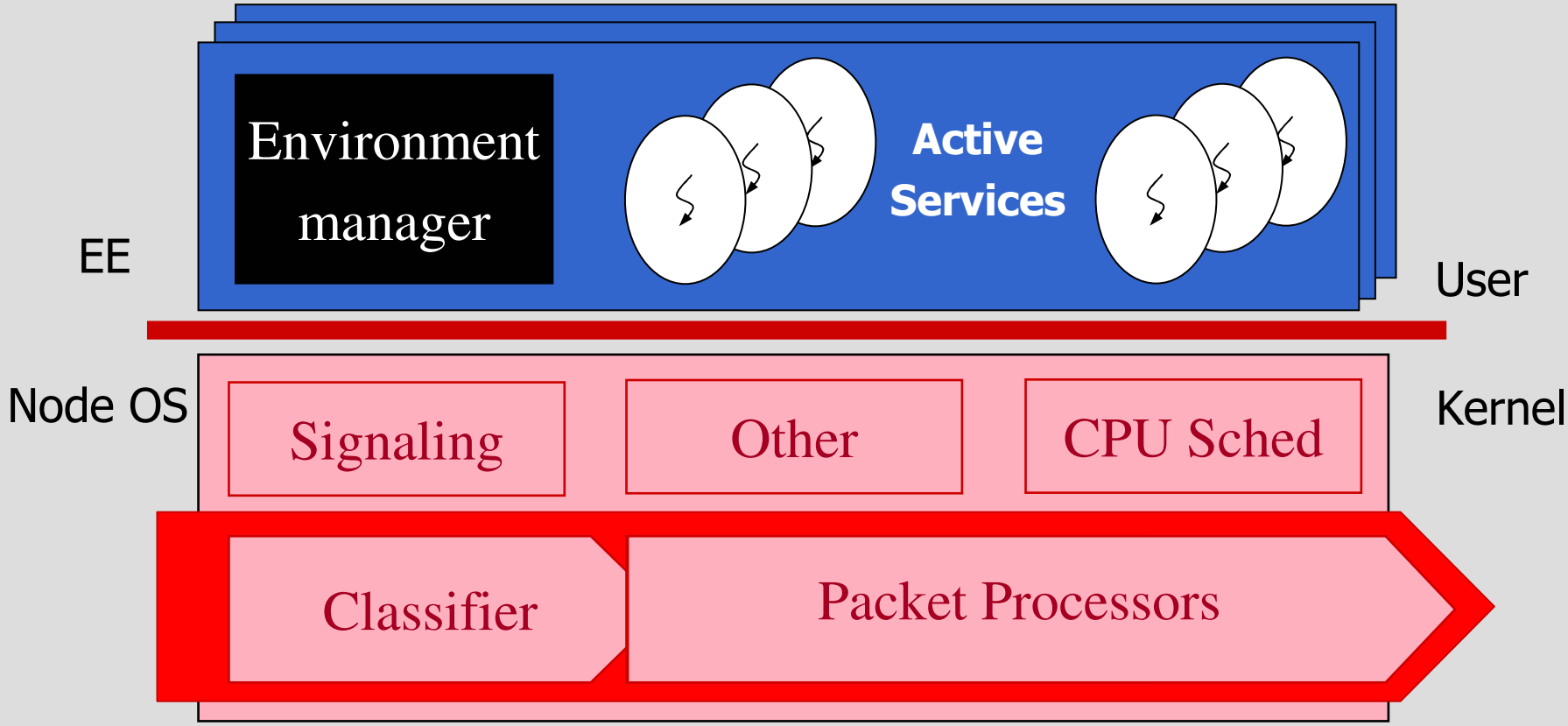
… more issues

- Level at which we program
  - Hardware, link layer, forwarding engine, IP layer, transport layer, applications
- Safety and robustness of operations
  - Fairness, liveliness, isolation
- Security - how secure must it be?
  - Who gets to program the network
- Composability
  - Can it be done? Is it worth the trouble?
- Granularity - of programming, of composing
- Killer apps - Is there one? Is one needed?

Overview

- Motivation, drivers and background
- The Pronto router
  - Programmable services
- New principle of timescales
- Evolvable router architectures
  - Do network processors matter?
- Reflection - what have we achieved?
- Where do we go from here?
- Summary

The Pronto Router

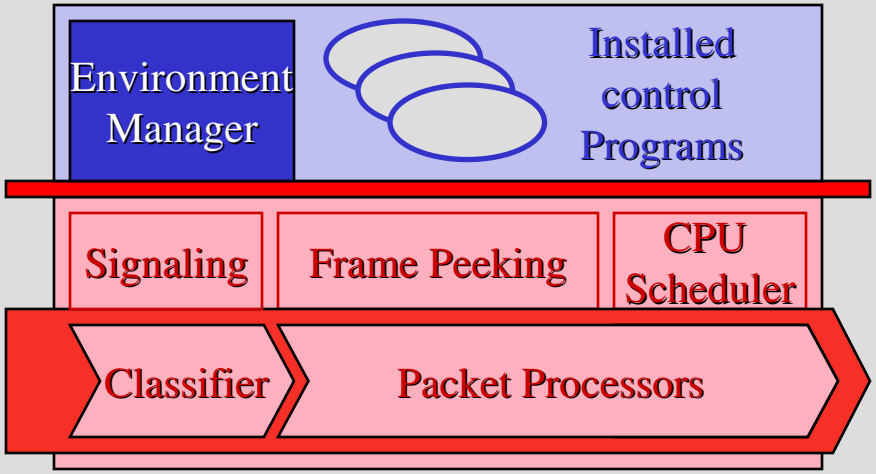


- Consistent with the DARPA Node OS framework
- Service programming provided in EE
- Router extensibility supported by packet processors

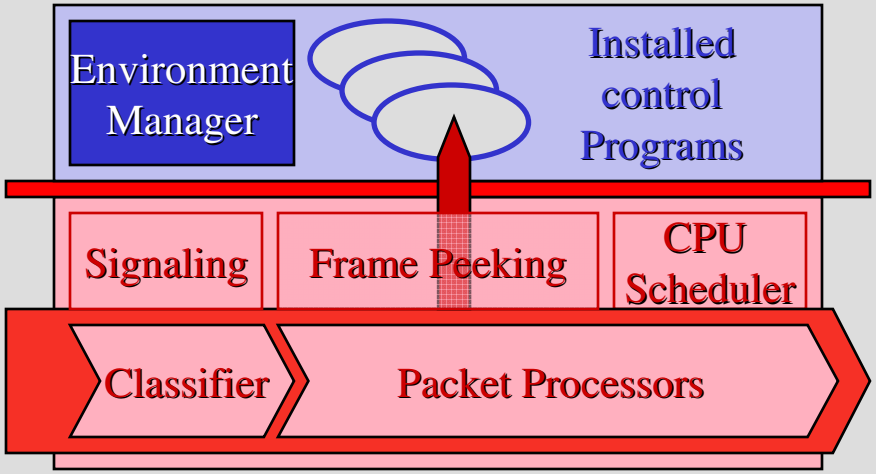
Service programming with Pronto

- Dynamic introduction into EE (user level)
  - Program using C++, Perl, Java
- Programmed by service providers
- Exploration in timescale of activity
- Four levels of programmability
  - Control Plane (optical switching, multicast)
  - Peeking into the data path (congestion adaptation)
  - Copying the data path (proxy caching)
  - In the data path (IPsec, transcoding, …)

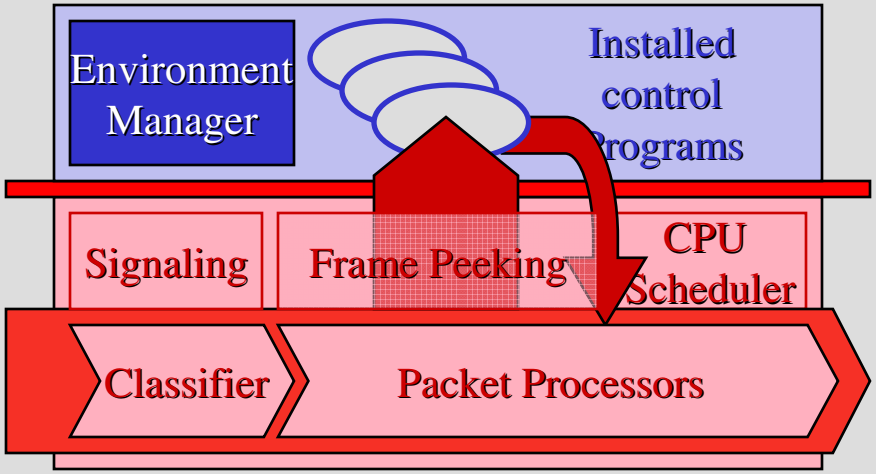
Four levels of programmability



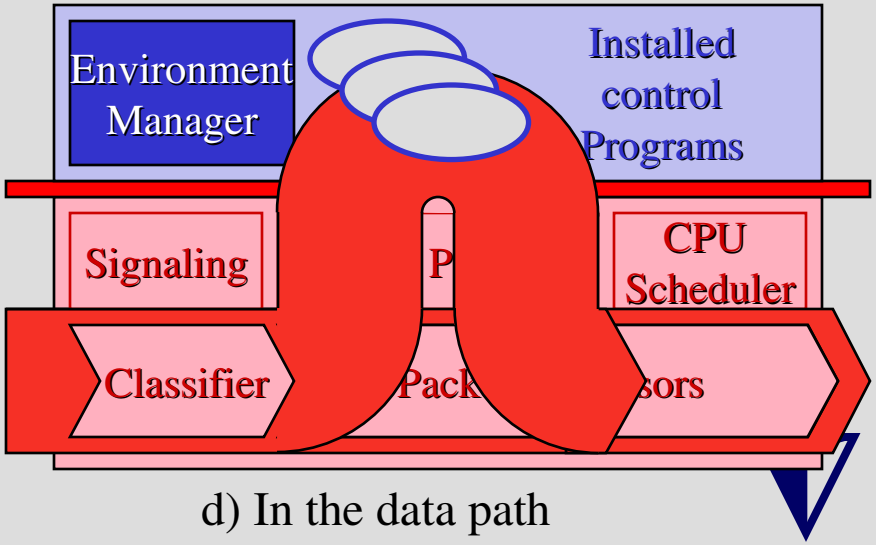
a) Control plane only



b) Peeking into the data path



c) Copying the data path

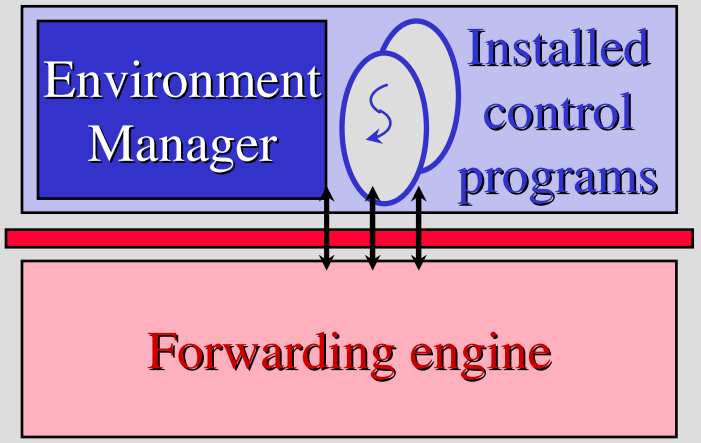


d) In the data path

Semantics matters

- Enhancement control (not true for in-data-path)
  - Not needed for correctness, nor for forwarding
  - Robust to non-cooperation
- Typed flows, rather than typed datagrams
  - Still acts at connectivity level or at data level
  - May assign, and reassign policies at any time
- Minimal impact on performance
  - The only essential work performed is forwarding
  - Control applied asynchronously to forwarding
  - Not essential to process every packet (“best-effort”)
- More predictable scheduling improves efficiency

Example - Selective Discard



- Congestion adaptation
- Assume MPEG stream
  - Packets part of I, P or B frames
- First packet requests control
- When the controller is activated
  - If the per flow Q is above some high water-mark
  - the controller discards P and B frames
- No synchrony, asynchronous activation
- Adaptation on different time-scale than forwarding

Example: Selective Discard

```
Selective_Discard()
    long peekLen := || application-level-header ||;
    (Qinfo, matches, signature) := query (myq, 0, peekLen);
    resultList result[ || matches || ] = nullresult();
    if Qinfo.Bytes > highWaterMark then
        for ( i=0; i < || matches ||; i++ ) do
            application-level-header packet :=  matches[i];
            if (packet.type != I-frame and Qinfo.Bytes > lowWaterMark) then
                result[i] := DISCARD; Qinfo.Bytes –= packet.length;
            endif
        endfor
    endif
    if result != nullresult() then response (myq, signature, result) endif
end Selective_Discard;
```

The Principle of Timescales

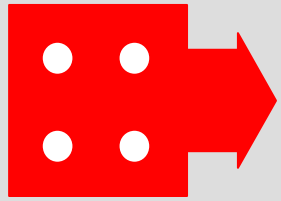
- Design principle for network control
- Push functionality to the highest layer/plane viable for the timescale of a given task
- Applied this to a number of problems
  - Congestion adaptation
  - Connection tracking adaptive firewall
  - Exception handling in optical networks
- Pushed hard to figure out ways to separate timescales

Overview

- Motivation, drivers and background
- The Pronto router
  - Programmable services
- The new principle of timescales
- Evolvable router architectures
  - Do network processors matter?
- Reflection - what have we achieved?
- Where do we go from here?
- Summary

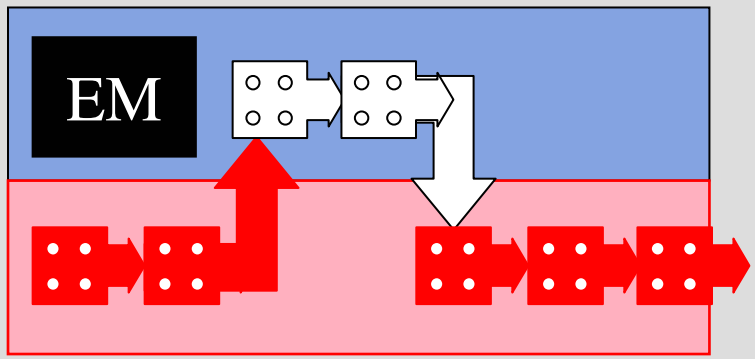
Extensibility of the Pronto router

- Packet processors – A new kernel abstraction
  - Provides programmability for hardware vendors
  - Intended for router providers - enhance facilities
- Simple abstract interface (C++)
  - New types implement this interface
  - Can do arbitrary processing
- Type specific service interface (SAPI)
  - Extends the programmable interface !!
- OO abstractions – C++ implementation
  - Additions maintain type invariants
  - Type specific interface extensions



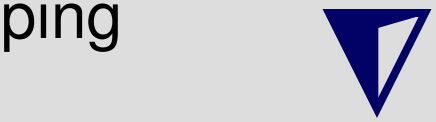
Paths – Chains of processors

- Chains of packet processors make paths
  - Assigned to a filter in the classifier
  - Single filter may map to multiple paths
  - Synchronous or asynchronous chain execution
- Paths can cross into user space
  - Abstraction maintained
  - Can avoid data copying
- May cross CPU boundaries
  - E.g. NP's on interfaces



Implementation

- All implemented in C++, in OO manner
  - New packet processors implement the abstract interface
  - Exploit C++ to the fullest – strong typing, inheritance, virtual functions, rtti, even exceptions (kernel level)
- New types extend SAPI - type specific interface
  - Implement late binding of system calls
- Services manipulate proxy objects
  - User level library – corresponding to types of pprocs
- Examples
  - Forwarding, Tunnel entry/exit, IPSEC tunnels, NAT, TCP splicing, dropping, QoS, restoration, snooping

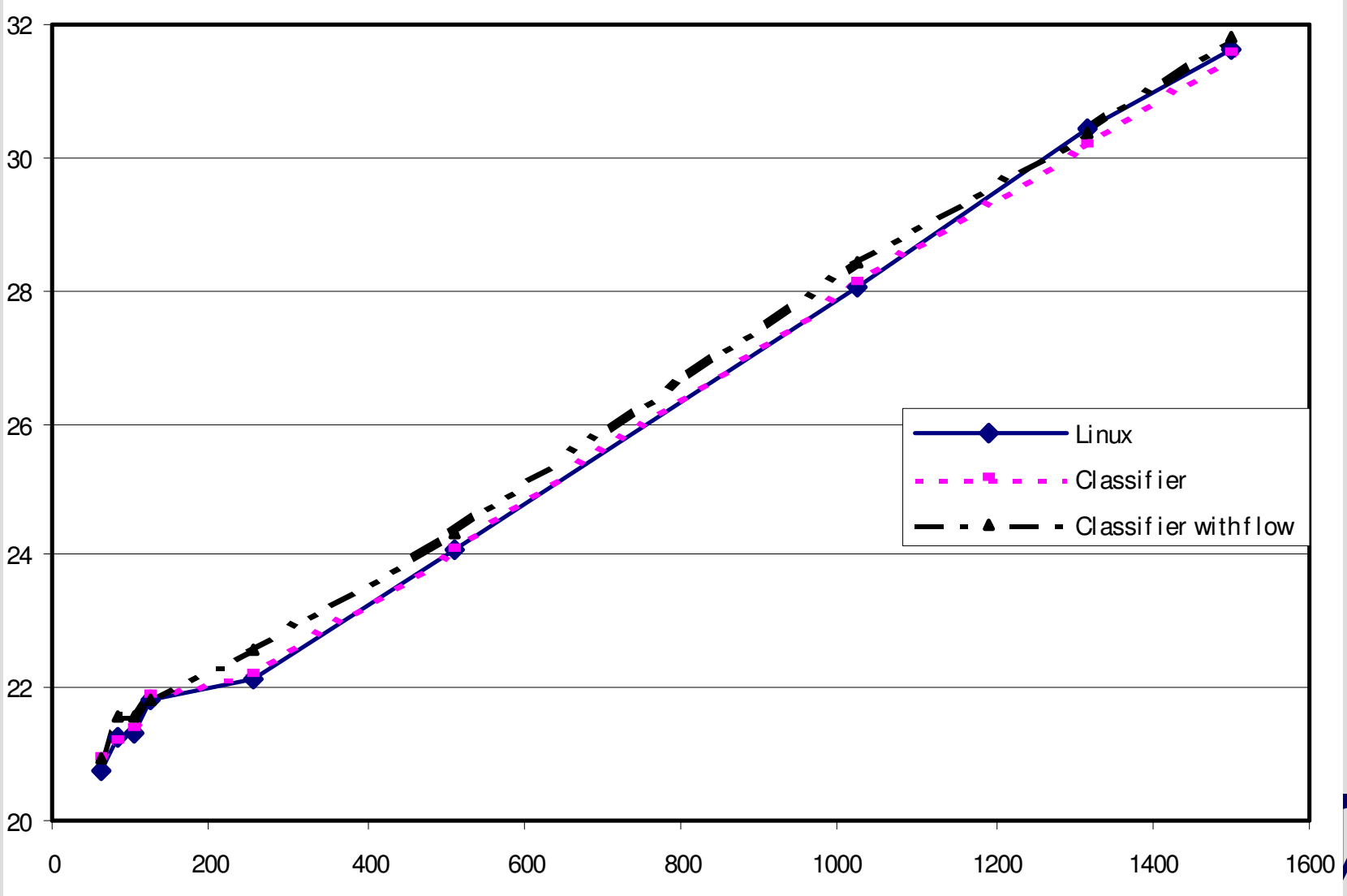


The pproc interface

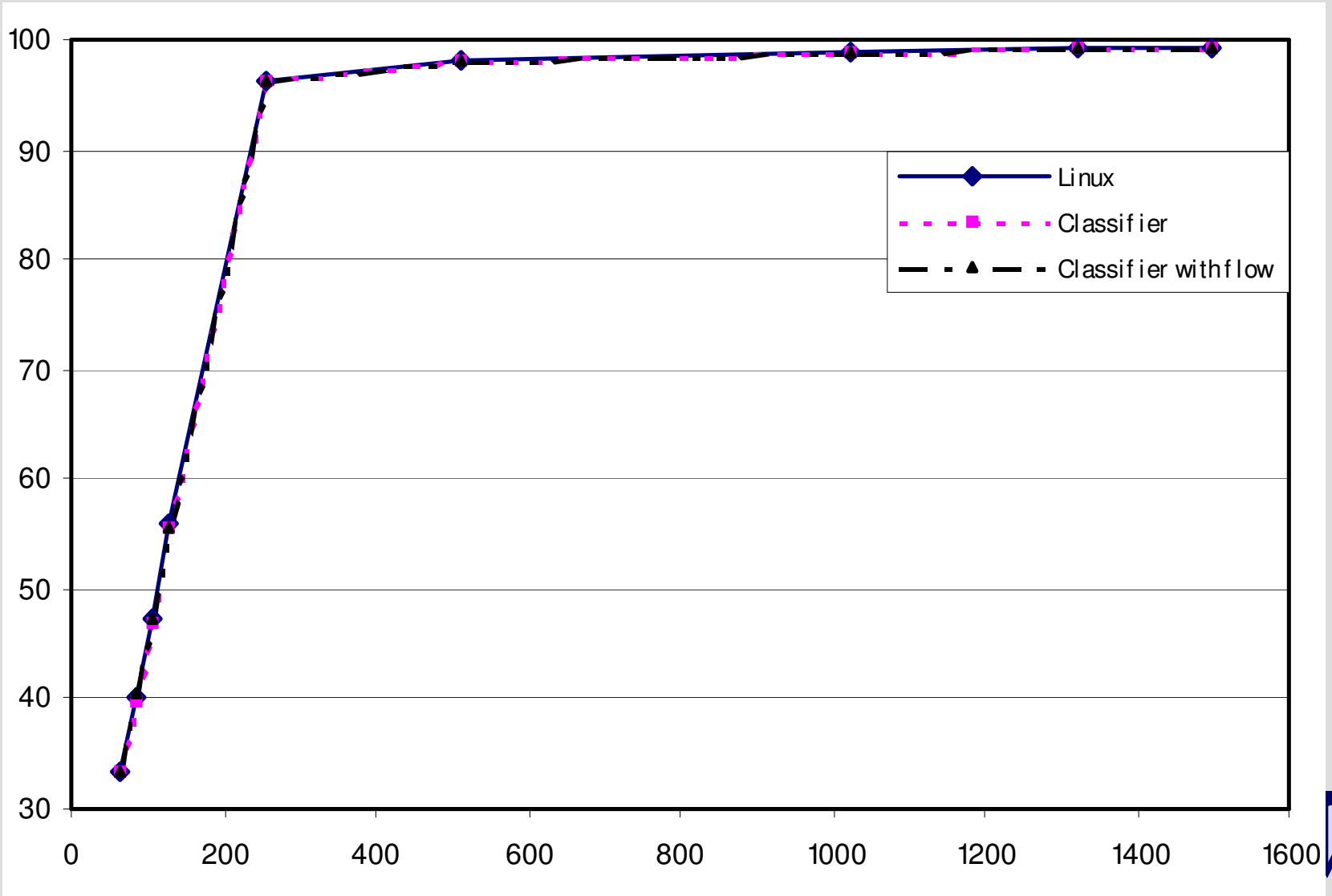
```
class pproc_iface {
public:
    pproc_iface() {}
    virtual ~pproc_iface() {}

    virtual int arrive(struct sk_buff *skb) =0;
    virtual int link_to(int pptype_id, int ppinst_id)=0;
    virtual const char *type_name()=0;
    virtual int destroy()=0;
}
```

Latency in μs vs. packet size



Throughput in Mbps



Pronto using network processors

- Have implemented pprocs on network processors
  - Abstractions remain valid
  - Implementation of classifier and basic pprocs
  - Still experimenting with how opaque we can make it to the service programs
- Network processors certainly do not enhance programmability
  - Practically impossible to program
- Goal to enhance the usability of network processors
  - help Intel provide common pprocs

Overview

- Motivation, drivers and background
- The Pronto router
  - Programmable services
- The new principle of timescales
- Evolvable router architectures
  - Do network processors matter?
- Reflection - what have we achieved?
- Where do we go from here?
- Summary

So what's the big deal?

- Time scale of programmability – NOT AN ISSUE
- Time scale of activity – Principle of timescale
  - Per packet timescale hard, 100 times that easy
- Model of programmability – Practical convergence
- Interfaces – a number of viable alternatives
  - Packet processors minimize this interface
- Program @ levels – services vs. extensible routers
- Safety and security – work to do, but not a showstopper
- Composability – Doable to some extent
  - Feature interaction problem remains.
- Granularity – not a critical issue

- **Proven that it can be done – no technical show-stoppers**

What remains undone

- No significant impact on router vendors
  - No significant router vendor offers an open programmable interface
  - May soon have to respond to threats from gateways
- No significant impact on the "Sigcomm" crowd

- Refocus on alternative ways to introduce services
  - Service deployment is focused overlays and P2P
- Application driven approaches
  - Incremental deployment a critical issue

Overview

- Motivation, drivers and background
- The Pronto router
  - Programmable services
  - Evolvable router architectures
  - Do network processors matter?
- The new principle of timescales
- Reflection - what have we achieved?
- Where do we go from here?
- Summary

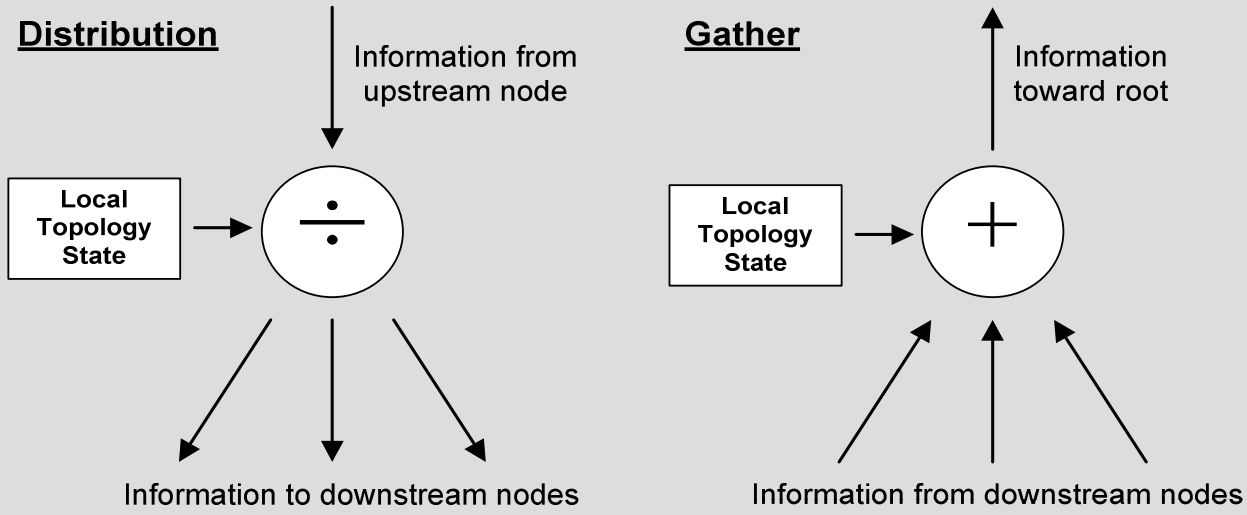
Distributed service level control

- Control plane across multiple nodes for the service
  - Exploit options to carry additional information
- Some of the overlay/P2P has nice ideas
  - Want to exploit them at control level
  - Still continue to get the benefits of network layer forwarding
- Particularly for multicast
  - Not a new multicast infrastructure
  - Only the information carried becomes richer
  - Selection of upstream hop becomes smarter, still local

Autonomic multicast service

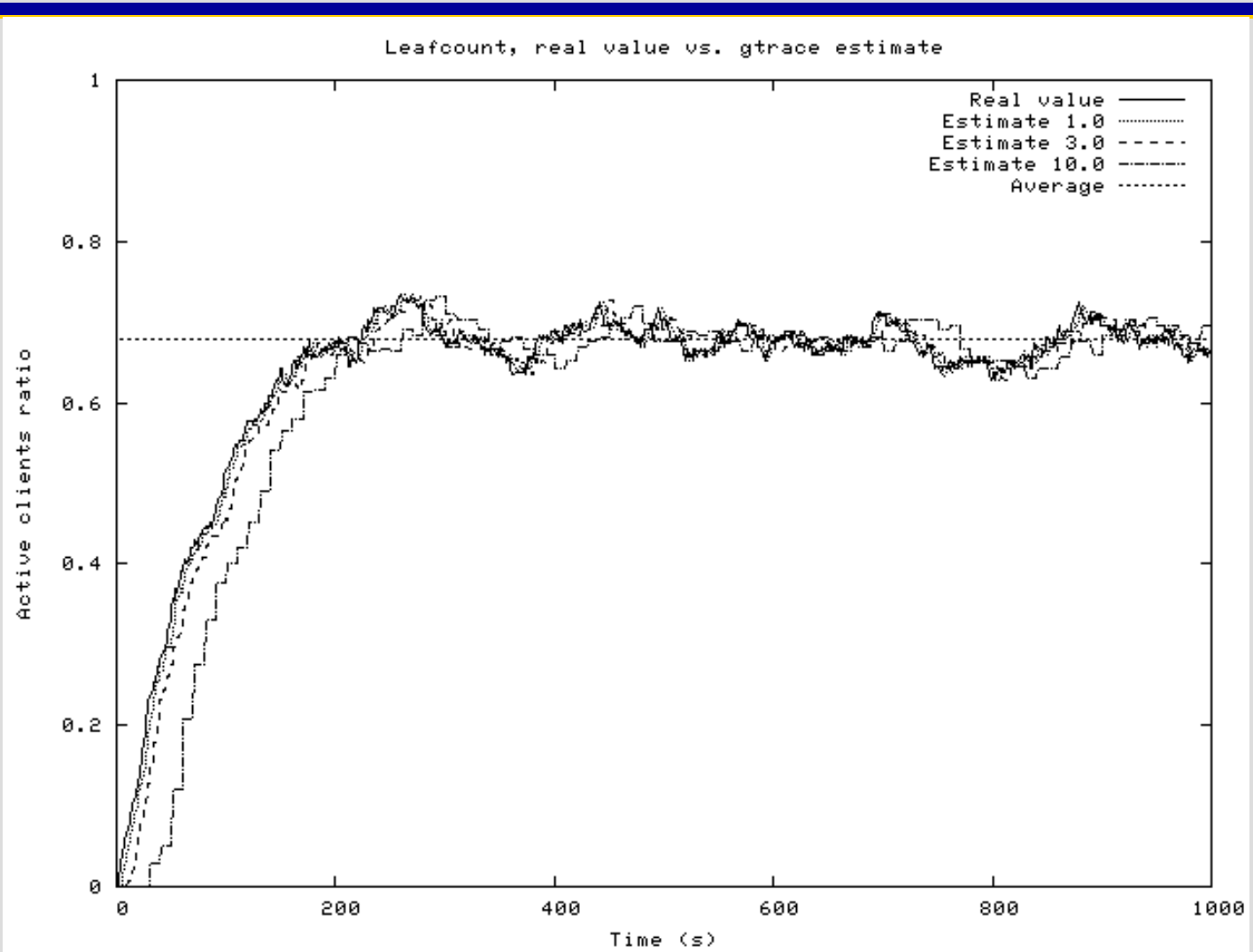
- Have built network layer SSM
- Monitoring protocol to collect distribution attributes
  - Combines a multicast/gathercast protocol
  - Collect basic properties of distribution trees
    - Height, weight, losses, etc.
- Each node in the distribution tree participates
  - Rapid tracking of tree properties
- Every node in the tree knows subtree stats
  - Each node recursively can act like a root of a subtree

Gtrace monitoring



- Local maintenance and collection
  - Collect information from local topology manager
- Information distribution
  - Propogates queries and information from a collection point to nodes of the collection channel
- Information gather
  - Gather information from nodes of the topology the root
- Can dynamically change the ÷ and + functions

Effectiveness of the monitoring



Autonomic adaptability

- Distribution tree recursively monitors performance
- When problems are detected:
  - Activate local retransmissions (reduce observed losses)
  - Increase priority of the data-stream (reduce latency)
  - Can dynamically introduce more functions
- Provides consistent service quality throughout the distribution tree
- Working on including recovery/restoration in this
- Example: TV station
  - Declares service level objectives (e.g. loss/delay)
  - Distribution tree autonomically tries to maintain service

The Routerless Internet

- Maybe the PlanetLab is the future
  - Smart end-systems using overlays over dumb IP forwarding network
- Increasing push for a cycle/pipe model
  - Nodes not offering cycles seen as dumb forwarders
- In fact we already see such a model
- All real routing points are becoming firewalls
  - Gateways and peering points
  - Active networking may be going mainstream
- Safety and resource management main issues

Summary

- Programmable networking is becoming mature
- Know how to provide service programmability
- Also how to provide router extensibility
- No technical show-stoppers

- Network management remains main issue
- Need more self-configuring/autonomic networks
- Increasing emphasis on overlays
- Self-configuring service/session level control plane
- See: netlab.ru.is