Towards High Speed Network Analysis

The NetVM Approach

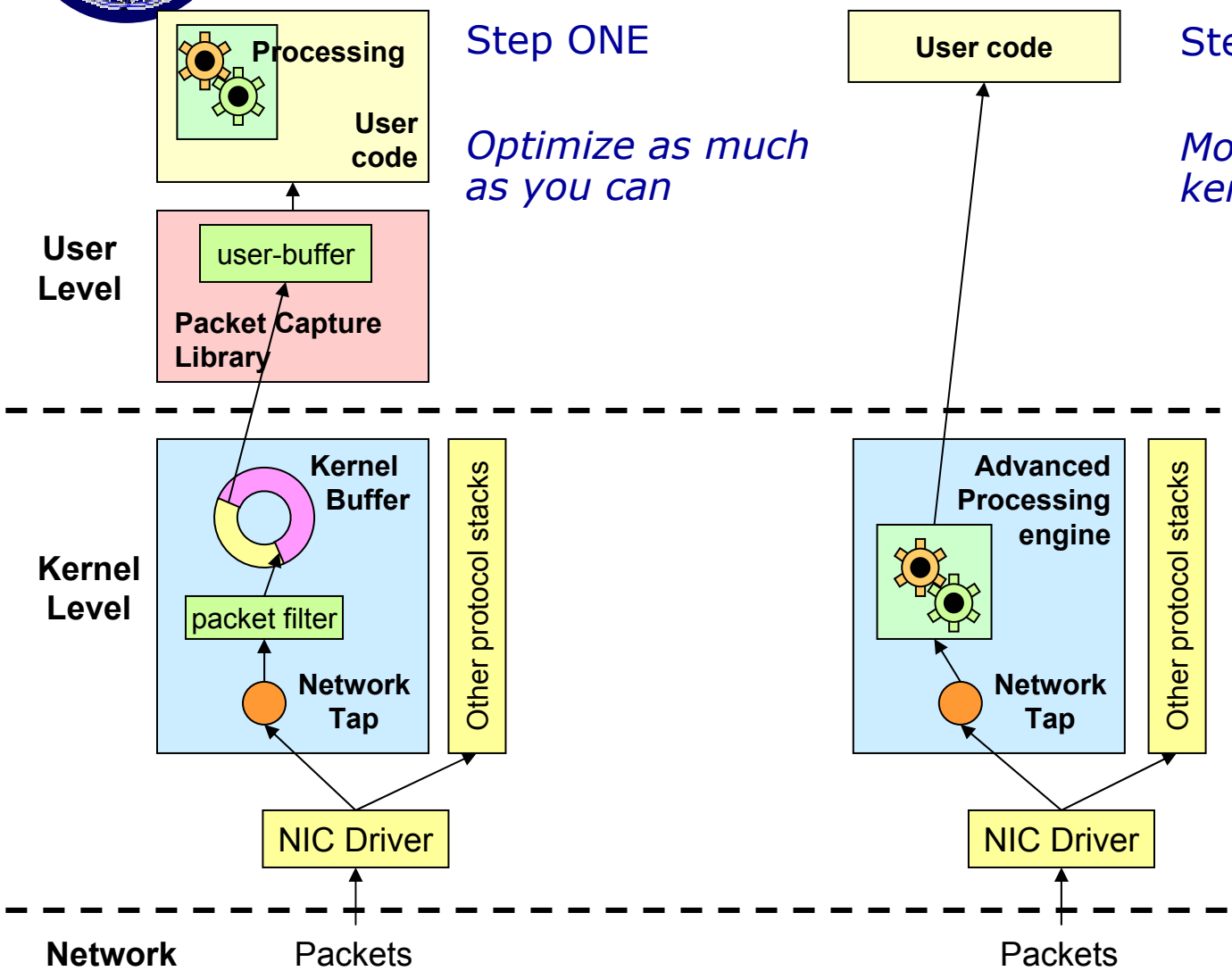
Fulvio Risso (fulvio.risso@polito.it)

Outline

- The path to High Speed Network Analysis includes two steps:
  - "Raw" performance
  - Clever components

- So, that's the outline:
  - How to increase performance in packet capture
    - In general, applications that need to process information contained in network packets
  - How to create smarter processing engines

The path for raw speed (1)



Step ONE

Optimize as much as you can

Step TWO

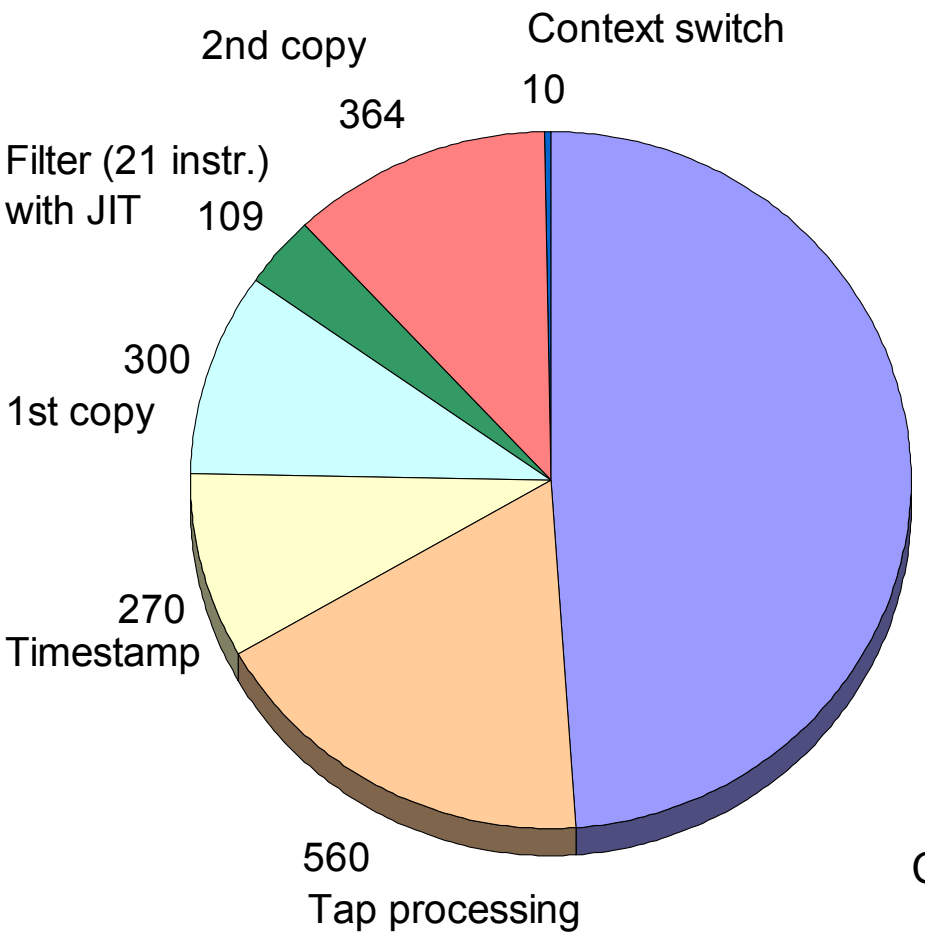
Move intelligence into the kernel

Decrease overhead when moving data around

Not suitable for some applications like packet capture

Remaining issues: interrupts and some kernel-related overheads

Let's talk about performances...



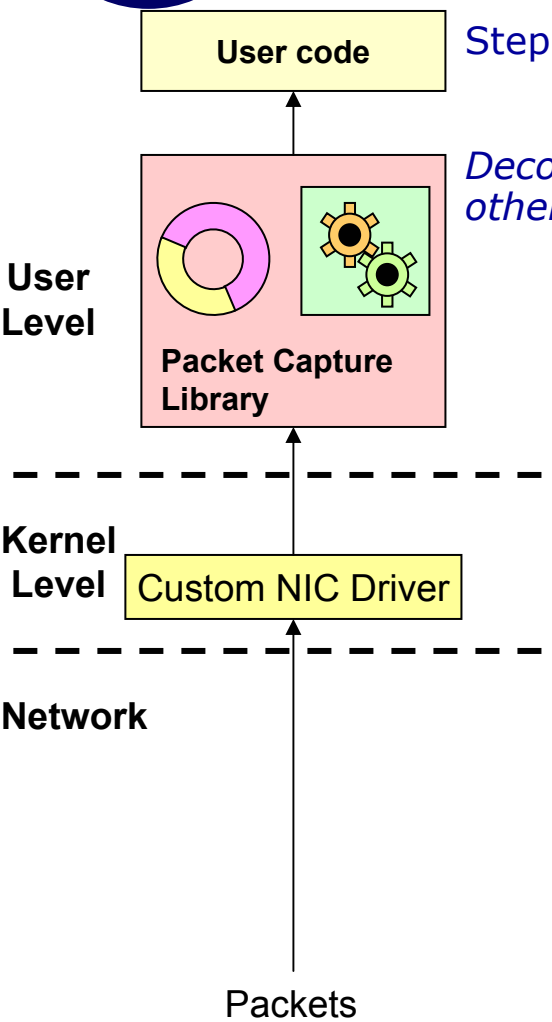
To increase performance should have:

- Hardware-based timestamp
- Avoid NIC driver and OS-related costs
- Avoid un-necessary copies
  (e.g. shared buffer)

Current Winpcap 3.0 overhead in clock cycles:

3164 clock cycles

The path for raw speed (2)



Step THREE

Decouple capture stack from other network stacks

Custom NIC driver

No longer able to support other protocol stacks (e.g. TCP/IP on the interface)

(Possible) intrusive modification of the operating system

Astonishing performances

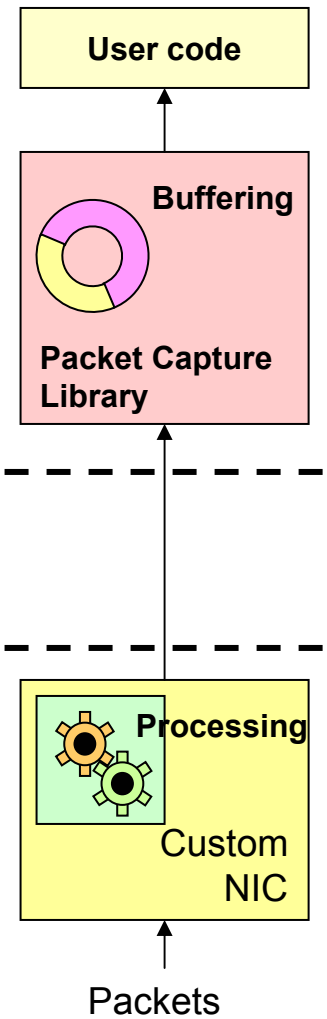
Capture on a Gigabit Ethernet

PCI bottleneck

Timestamp precision

Processing

Either in Kernel or User space (better)



Step FOUR

Create smarter NICs

Hardware processing

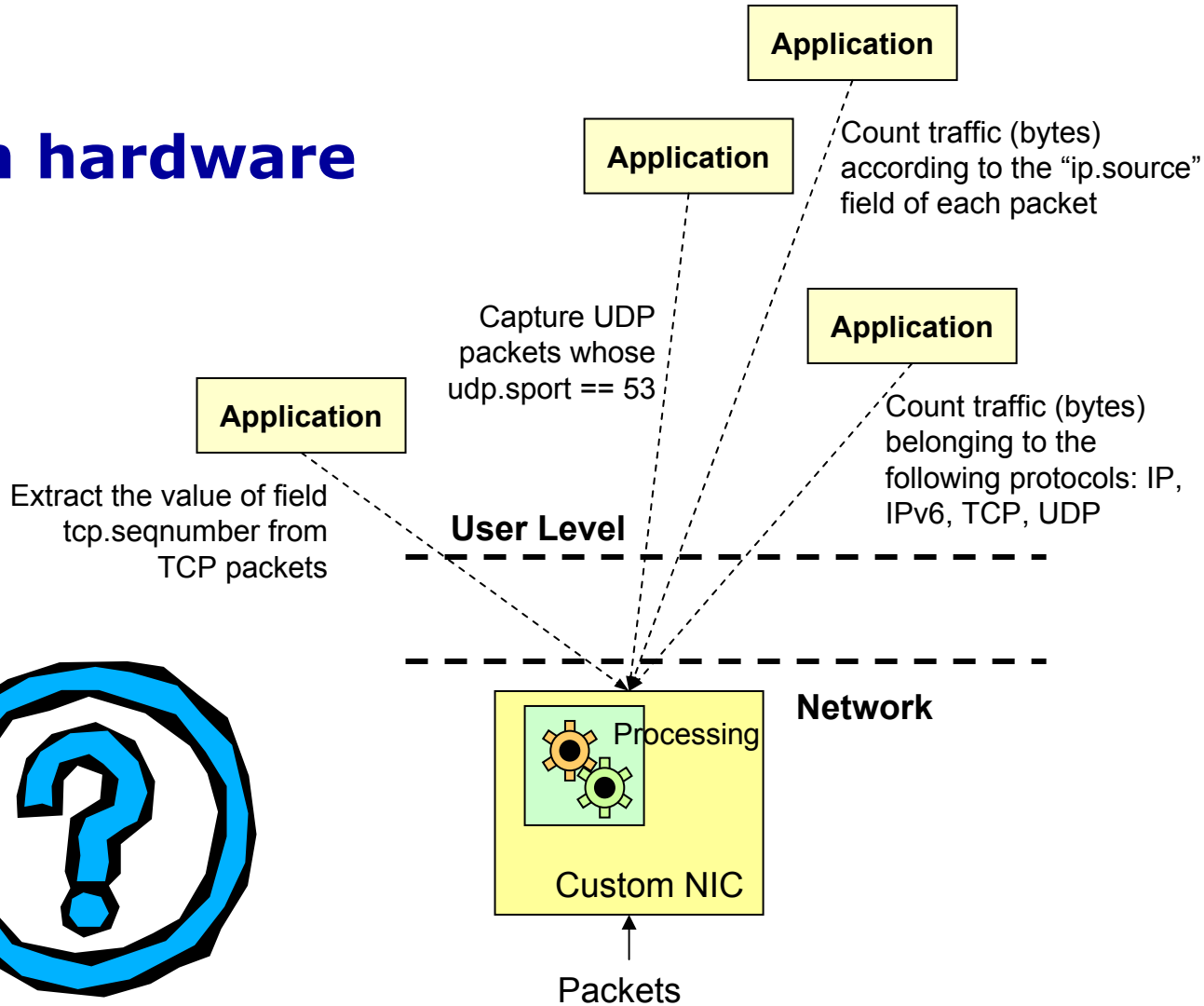
Avoid PCI bus bottleneck (not applicable for "capture all" applications)

Timestamp precision

Need advanced mechanism for customizable processing

So, the ultimate step is...

... processing in hardware



but... which kind of processing?

Which kind of processing?

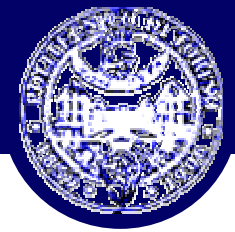
What do we want to implement in hardware?		
A set of complete applications: - Firewall - NAT - Traffic monitor (e.g. RMON) - …	Simple packet processing: - Packet Filtering - Packet Classification - Field Extraction - ...	A s… generic application: - Firewall - NAT - Traffic monitor (RMON) - …
What do we need to accomplish this goal?		
We need a card with a general purpose on-board CPU	We need a card with a Network Processor on-board	We need a card with an ad-hoc ASIC on-board
Which are the tradeoffs?		
+ Versatility - Speed General-purpose CPU are not the best choice for packet processing	+ Versatility + Speed - Not suitable for complex tasks Something should be done by the hosting workstation Possible PCI bus bottleneck	+ Speed - Versatility - Cost
Why don't you use a PC?	*Are network processors mature enough?*	*Do we want to create a new network appliance?*

Other options?

- FPGA
  - Mixture between Network Processors (reprogrammability) and ASICs (speed)
  - We know that many people are able to play with VHDL, but… how many network managers / security experts / etc are able to play with it?
    - They are usually able to use applications or write C code
  - May be an interesting option anyway
- Offloading engines
  - E.g. Crypto, classification, etc.
  - Are these functionalities enough for reaching high speed?
  - How difficult is to interact with them from user-applications?
  - What about if we need other functionalities (i.e. new engines)?

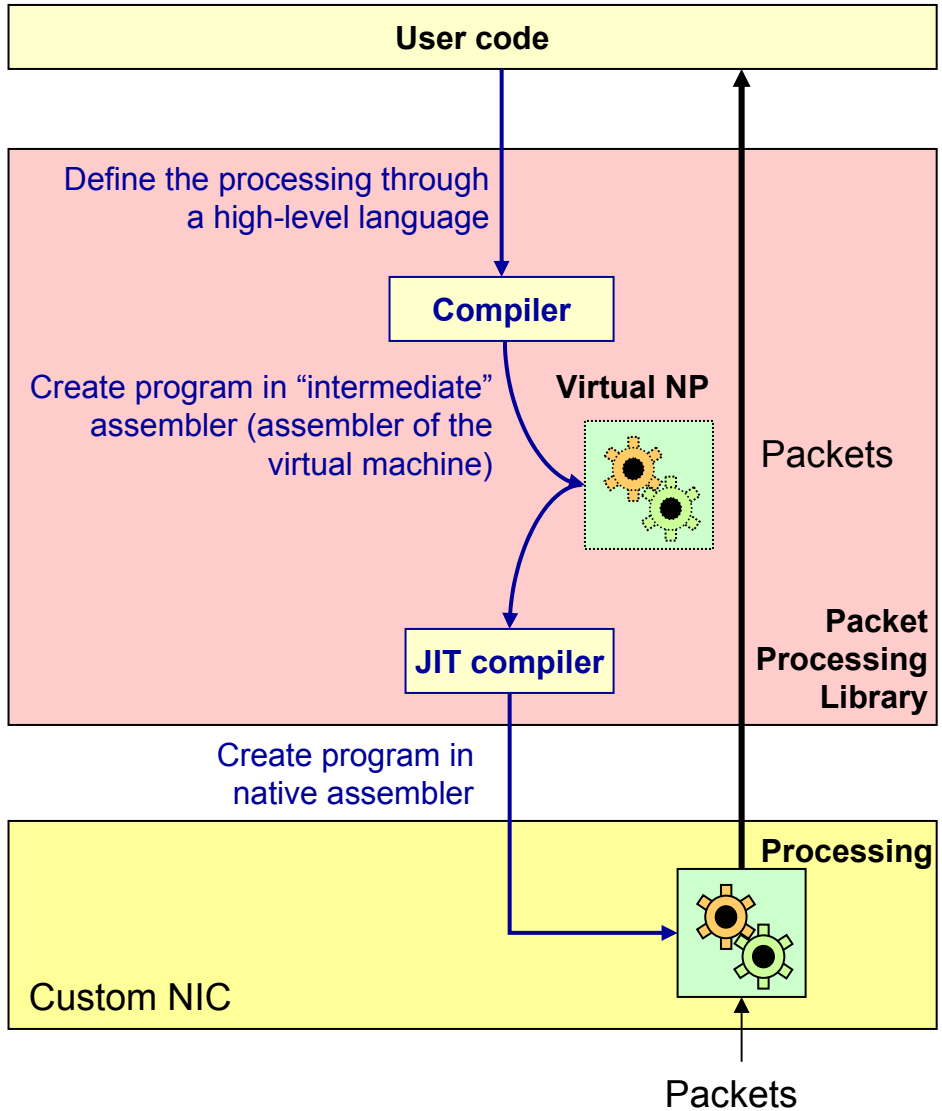
The best compromise...

- The Network Processor approach seems to be the best option:
  - + Engineered to operate on network packets
  - + (Potentially) programmable
  - + Fast
  - + We can exploit advances in silicon technology
  - + Usually it includes several “micro-engines” that can work in parallel / pipeline
- But:
  - – Difficult to program (do we need assembly?)
  - – Not very mature
  - – How can we deal with different NPs?
  - – Is there any simple way to exploit their intrinsic parallelism?



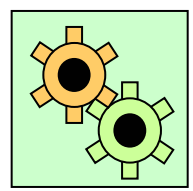
... is a Virtual Network Processor!

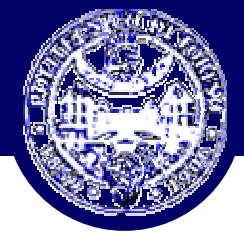
- Optimized to operate on network packets
- Programmable (programs can be customized)
- Fast
- Lightweight
- Exploits advances in silicon technology
- Several micro-engines can work together transparently (from user perspective)
- User programs can benefit from hw resources thanks to the JIT compilation
- Adaptable to different hardware (only?) architectures



Esisting alternatives: BPF (1)

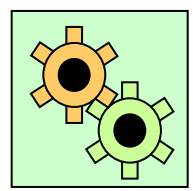
- Berkeley Packet Filter
- Widely used (embedded in libpcap, WinPcap and many operating systems)
- Register-based
  - Very simple: it uses only one register, which makes a Just-in-Time compiler easy to do
  - In general, register-based architectures are difficult to map efficiently in case the target architecture does not have enough registers





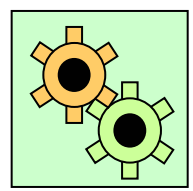
Esisting alternatives: BPF (2)

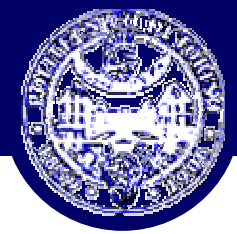
- Only one “processing engine” supported
  - No transparent support for multiple processing engines (in parallel or pipeline)
- Limited instruction set (e.g. backward jump)
- No support for “coprocessors” (e.g. hw components for doing hashing, encryption, etc)
- Works well for packet filtering (but *ipfw2* does not use it)
- Other solutions are similar to BPF



Esisting alternatives: Click

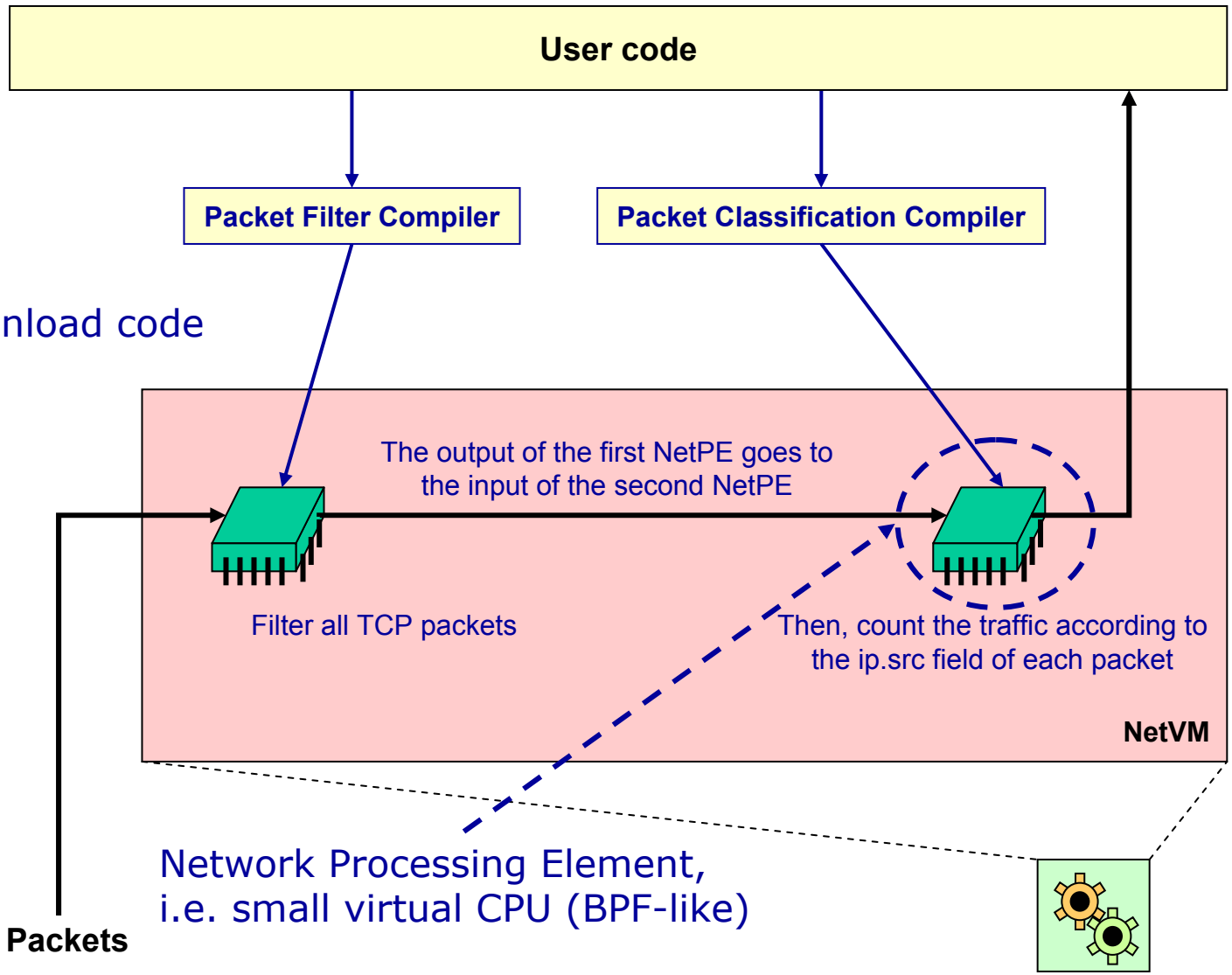
- Click (MIT)
  - Not really a "virtual network processor"
  - Modular and extendible router architecture
  - Listed here just because people tend to ask "what about Click"?
- Too many differences with the Virtual NP approach
  - Very complex components (e.g. IP forwarding, etc.)
  - Components cannot be programmed from user space
    - Does not have an "assembler"
    - It is a set of "pre-built" components
    - Cannot be mapped on existing hardware accelerators for specific functions
  - Does not support "remotization" (see later)
- Click "*does*", a Virtual NP "*allows you to do*"





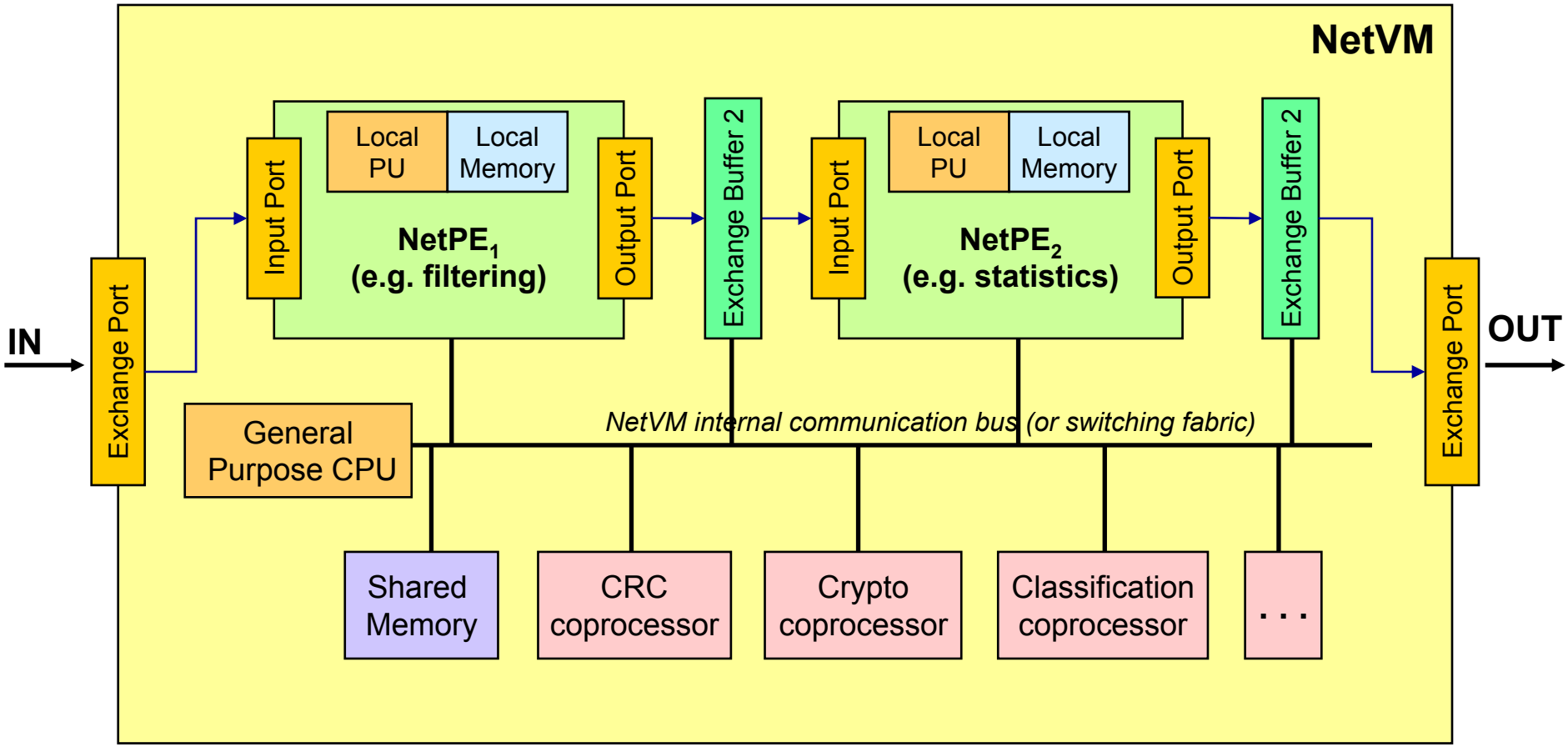
The idea: Network Processing VM

1. Instantiate PE
2. Connect PE
3. Compile and download code
4. Start execution

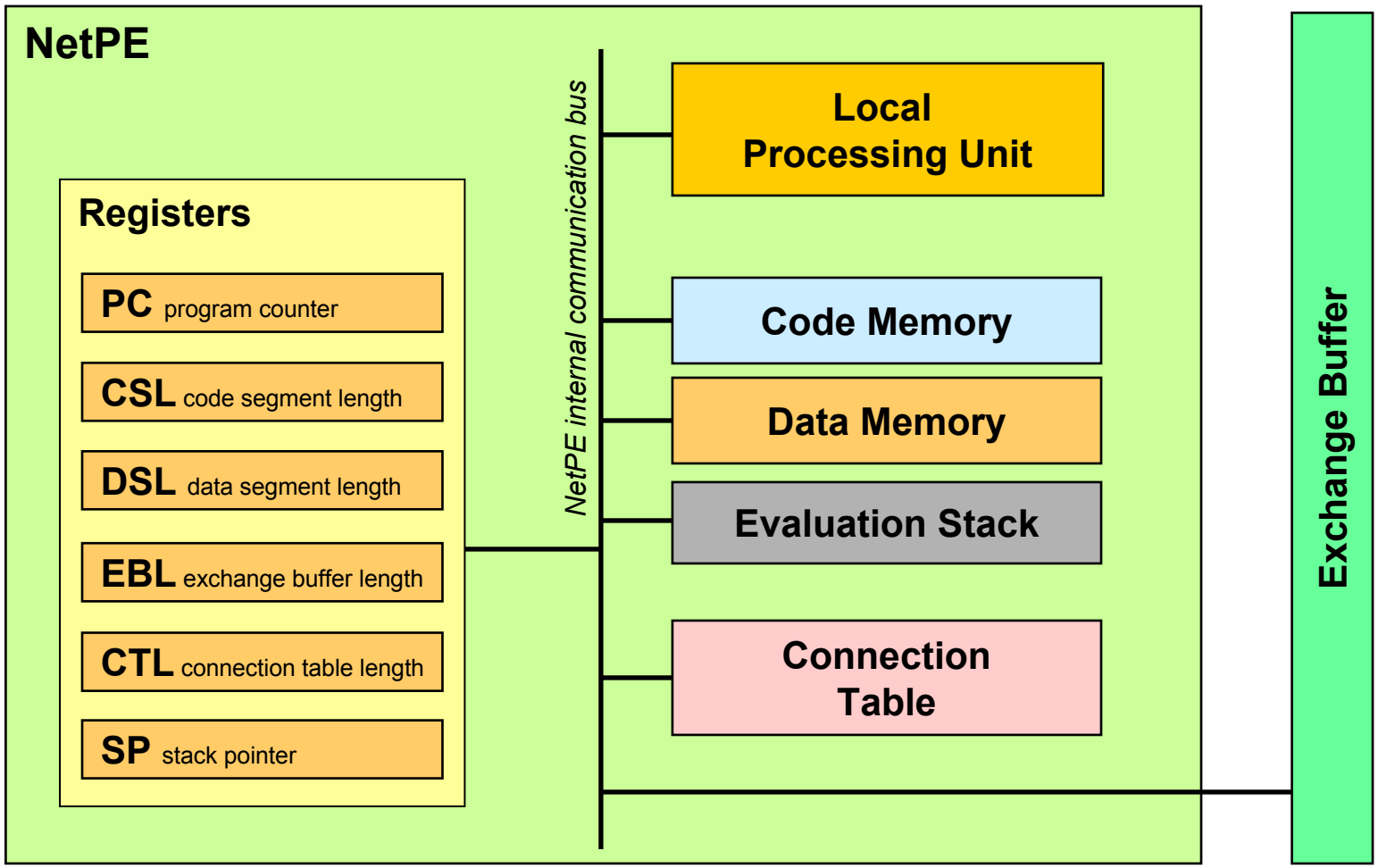


NOTE: the JIT compiler has been omitted for clarity

Network Processing Virtual Machine



Network Processing Element

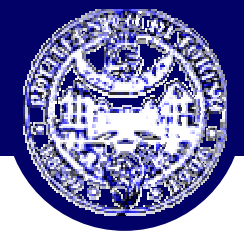


Some numbers...

	Berkeley Packet Filter (Win32 implementation)	NetPE
Performances	295 CPU ticks[1]	990 CPU ticks[1]
Lines of code	370[2]	1460

Tests run on a PC Dual Xeon, 2GHz clock

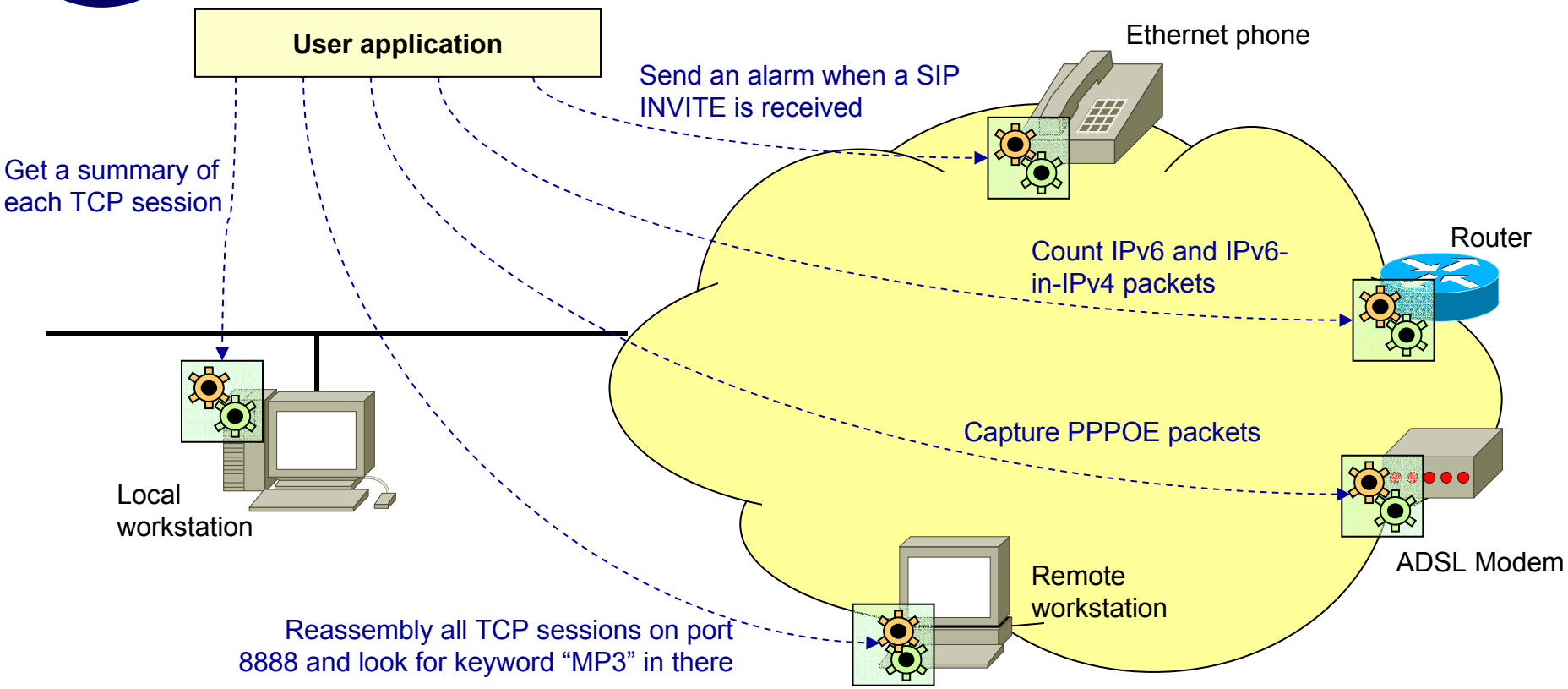
[1] 16 assembly BPF instructions against 36 assembly NetPE instructions (due to the several push/pop operations needed on a stack-based machine); no JIT compilation involved.
[2] In case of the BPF implementation, the JIT compiler counts for additional 650 lines of code.



Let's move a step forward

- If we want to play with remote devices, which kind of data can we bring home?
  - Network packets (thanks, WinPcap)
    - Only on PCs and Cisco Catalyst 9000
  - Aggregate traffic statistics
    - RMON, NetFlow
      - Are these suitable for appliances like ADSL modems?
      - How much do these cards cost?
      - Can you customize the type of statistics you want?
- What about if we want to:
  - Do more (not only statistics or packet capture)
  - Be open to the future?

The NetVM can be a solution!



Let’s implement the Virtual Network Processor in remote devices!

Remotizing the NetVM (1)

- Allow to create small programs that can be downloaded anywhere
  - Like *Java applets*
    - So, why can't we use Java?
      - Java has several features we do not need, while it does not have features we need
  - A remote "management console" can control their behavior

Remotizing the NetVM (2)

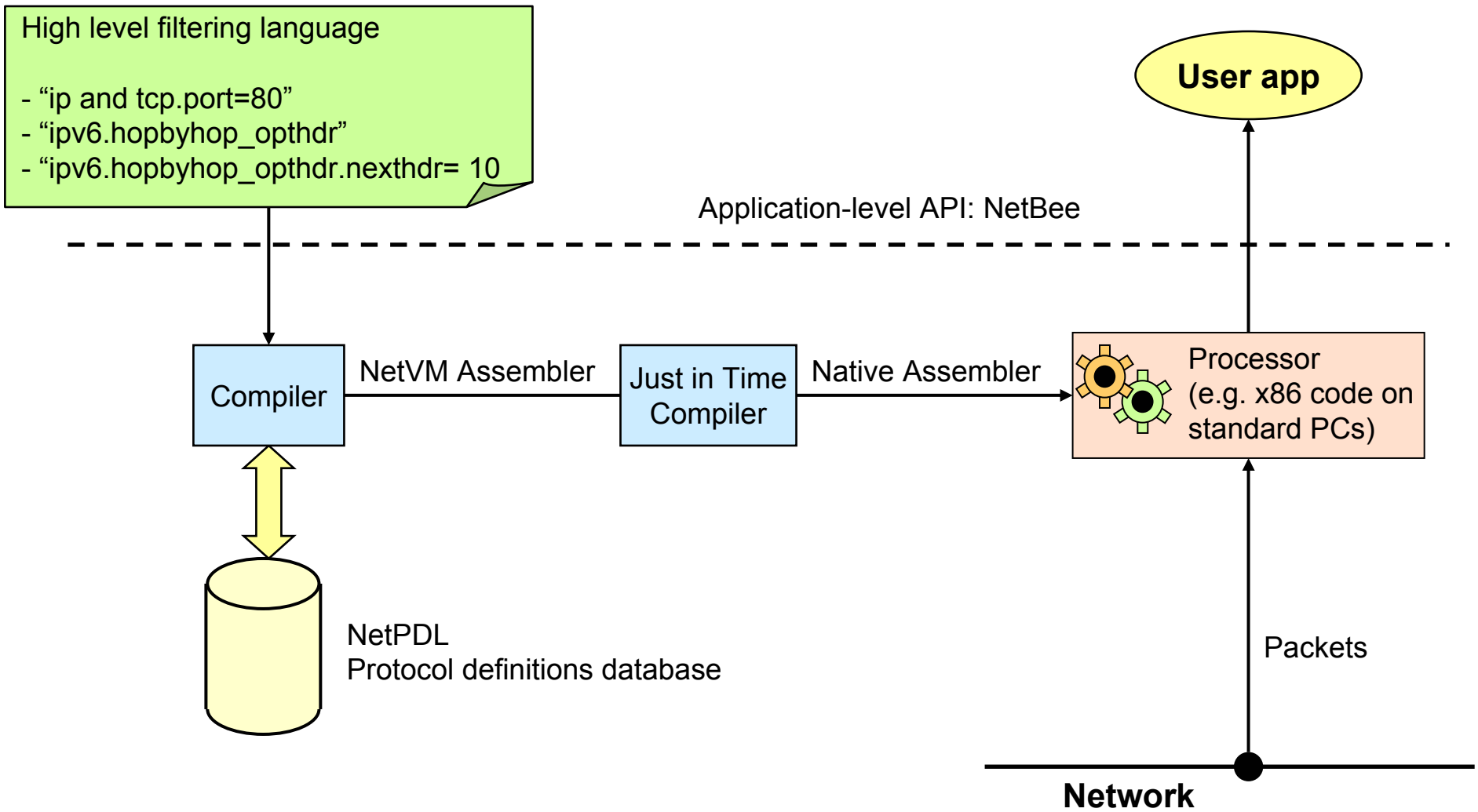
- Potentially, the network can become *programmable* (at least in some components)
  - We can have our firewall running on remote devices
  - We can customize generated alarms
  - We can compute custom statistics
  - Etc...
- NetVM and Active Networks
  - The code is not in network packets
    - It is downloaded through a “control channel”
  - Code is downloaded / managed from network admin
    - Limited security issues, no need to control access of resources
  - NetVM is feasible, Active Networks (probably) not

New issues due to the remote interaction

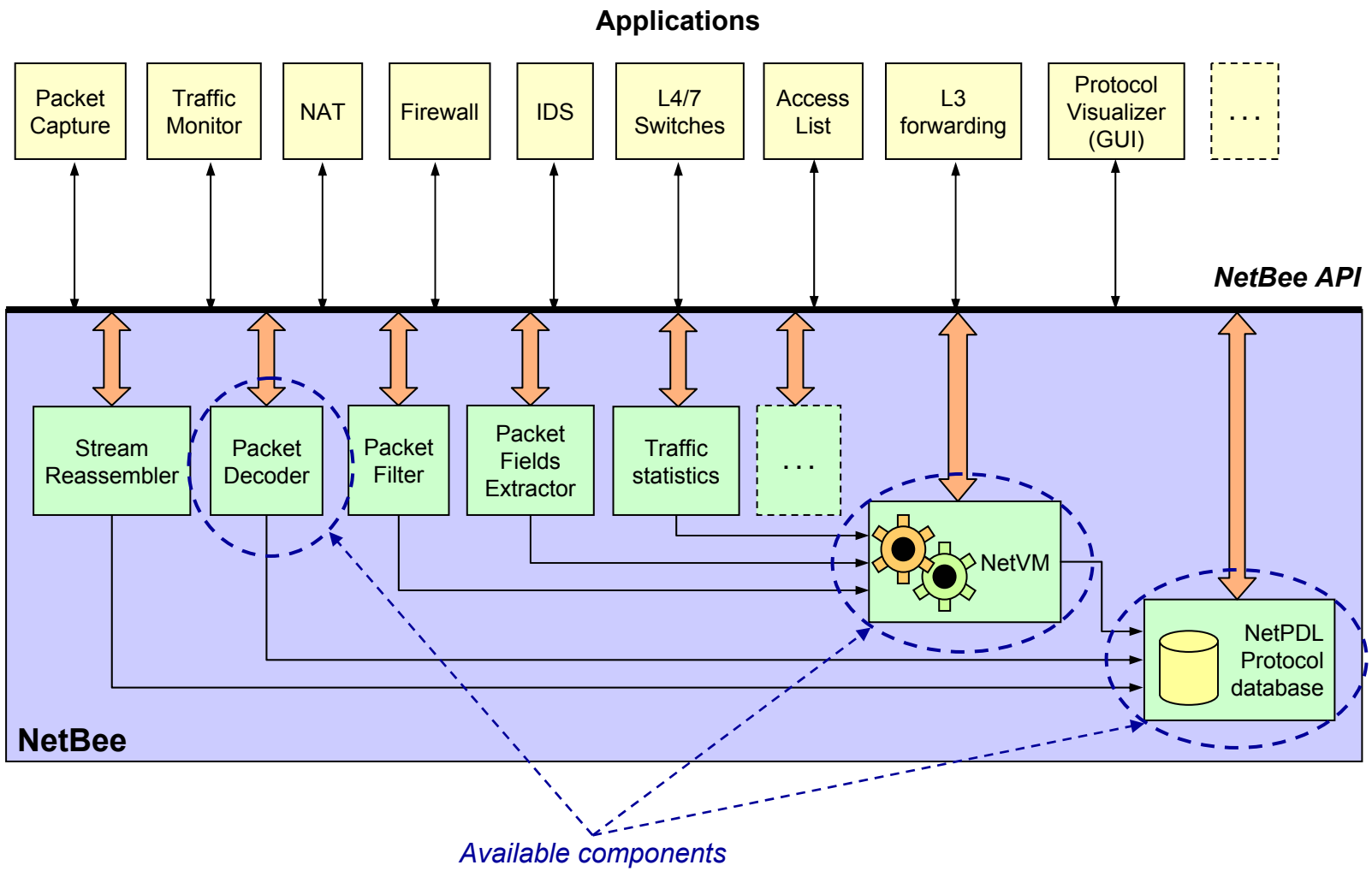
- To make the NetVM remote we need some new components
  - We need an extensible transport protocol to send commands to the NetVM and to get data back
  - User application must behave in the same way when the NetVM is local or remote

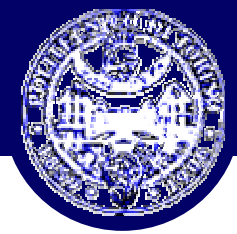
```
NetVMHandle= NetVMCreate("host.foo.bar");

/* some other code goes here */
NetVMHandle->DownloadCode(code);
Results= NetVMHandle->GetResult();
```

Putting the pieces together

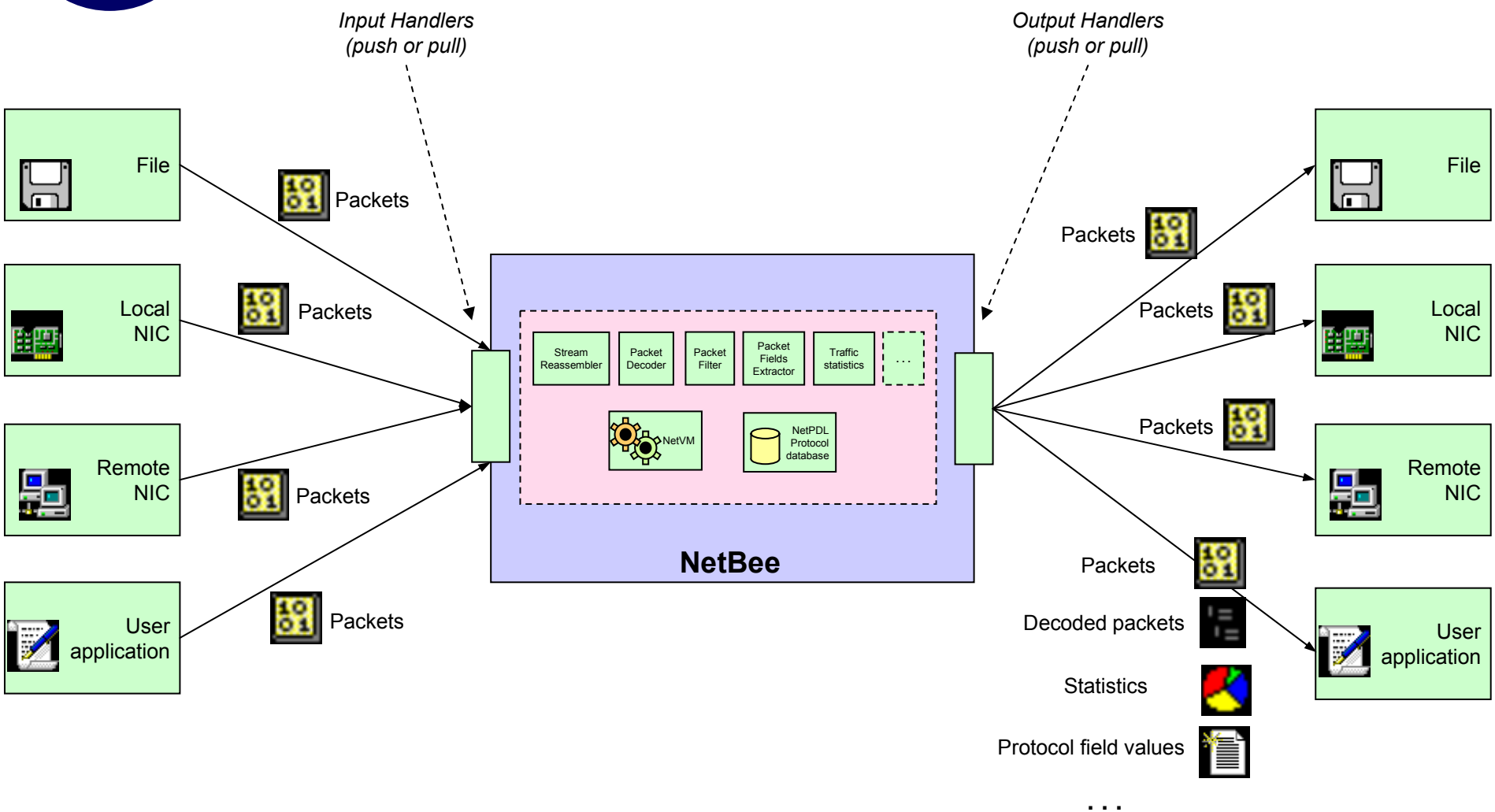


The NetBee Library: Control Path

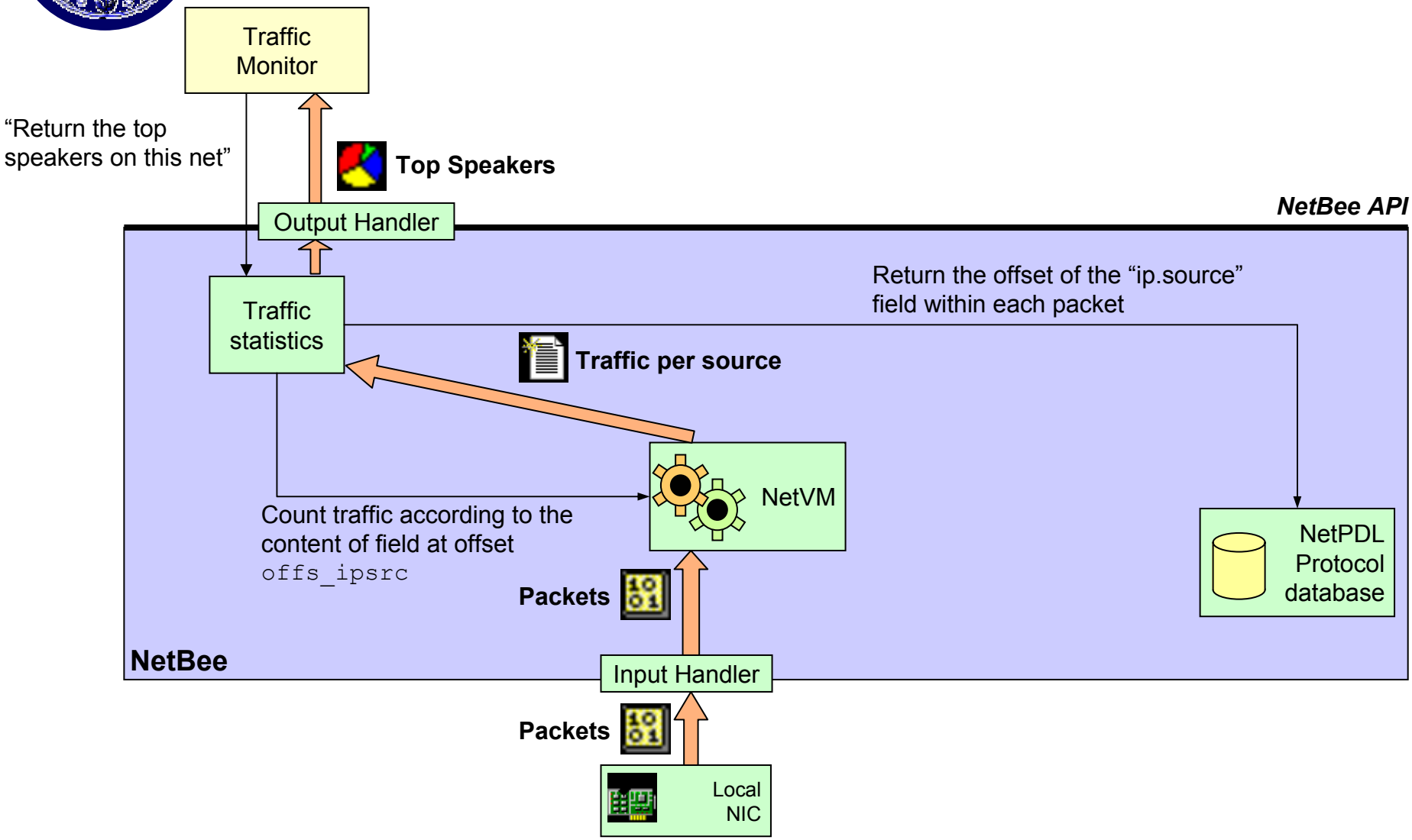




The NetBee Library: Data Path



NetBee: example of use



NetBee status

- New generation packet (manipulation, handling, etc.) library
  - Allows to program a NetVM
  - Hide the location of packets source (i.e. packet tap) and transfers processed data to the caller
- Some functionalities exist right now
  - Packet Decoding
  - NetPDL
  - NetVM (rather primitive)
    - NetPE (more stable)
- Still work in progress
- Both users and developers wanted

Conclusions

- Raw performance
  - This goal can be considered somewhat reached
    - Experimental optimized drivers exist
    - Commercial cards (DAG)
- More intelligent components
  - NetVM
- Interaction between user applications and NetVM
  - NetBee
    - Users are not expected to write assembly programs for the NetVM

Questions?

Thanks for your attention!