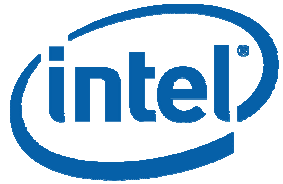


# Fast Prototyping of Network Data Mining Applications

Gianluca Iannaccone
Intel Research Cambridge

# Motivation

- Developing new network monitoring apps is **unnecessarily time-consuming**
- Familiar development steps
  - Need deep understanding of data sets (including details of the capture devices)
  - Need to develop tools to extract information of interest
  - Need to evaluate accuracy and resolution of data (e.g., timestamps, completeness of data, etc.)
- ...and all this happens before one can really get started!

# Motivation (cont'd)

- Developers tend to find shortcuts
  - Quickly assemble bunch of ad-hoc scripts
  - Not **"designed-to-last"**
  - Well known consequences
    - → hard to debug
    - → hard to distribute
    - → hard to reuse
    - → hard to validate
    - → suboptimal performance
- End result: many papers, very little code

# Can we solve this problem by design?

- Yes, and it has been done before in other areas.
- Define declarative language and data model for network monitoring
- What is specific to network measurements?
  - Large variety of networking devices (i.e. potential data sources) such as NIC cards, capture cards, routers, APs, ...
  - Need native support for distributed queries to correlate observations from a large number of data sources.
  - Data sets tend to be extremely large for which data shipping is not feasible.

# Existing Solutions

- AT&T's GigaScope
- UC Berkeley's TelegraphCQ and Pier
- Common approach (stream databases):
  - Define subset of SQL adding new operators (e.g., 'window' for time bins of continuous query)
  - Gigascope supports hardware offloading by static analysis of the GSQL query

# Benefits and Limitations

+ Decouple what is done from how it is done.
+ Amenable to optimizations in the implementation
- Limited expressiveness.
- Need workaround to implement what is not in the language losing the advantages above
- Entry barrier for new users is relatively high.
- Existing solutions not designed with a variety of devices in mind (mainly packet capture)

# Alternative Design: The CoMo project

- Users write “monitoring plugins”
  - Shared objects with predefined entry points.
  - Users can write code in C or whatever they like that can generate the shared objects.
- The platform provides
  - one single, extensible, network data model.
  - support for a wide variety of network devices.
  - abstraction of monitoring device internals.
  - enforcement of programming structure in the plug-ins to allow for optimization.

# Design Concepts

- Network Data Model
  - or, “how to find the data”
- Programming Model
  - or, “how to process and manipulate the data”
- Hardware Abstraction and Data Management
  - or, “how to optimize for performance”

# Network Data Model

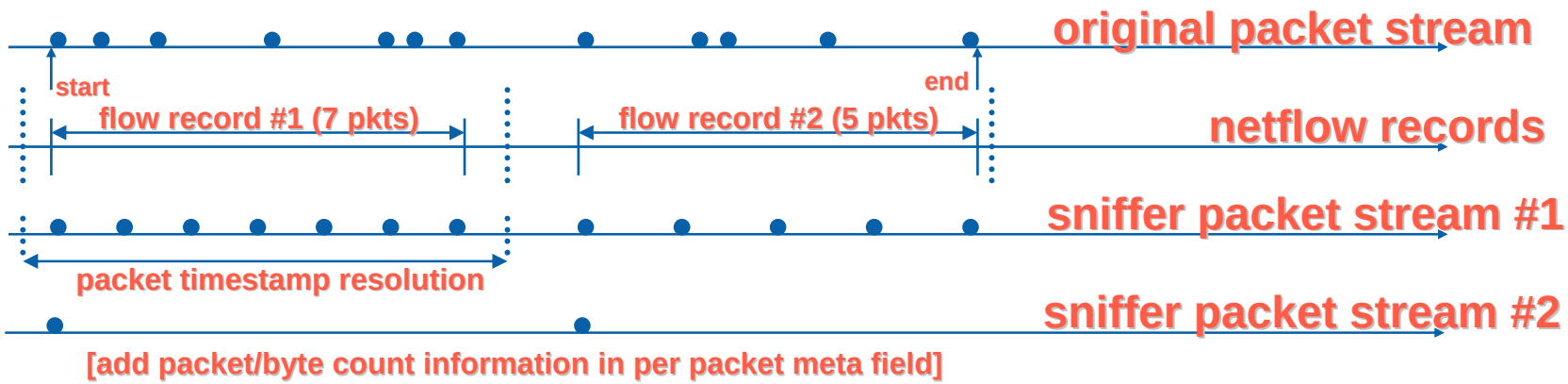
- Unified data model with quality and lineage information.
- Allows the definition of ad-hoc metadata (by users)
- Starting point is the IP packet
  - Add other protocol headers (MAC, transport layer, etc.)
  - Add other information that is capture device specific (e.g., PHY information, RF information, routing information)
  - Add per packet meta information (e.g., flow-level information) and per stream meta information (e.g., accuracy of timestamps)
  - Allow for specifying new fields by name in any packet (e.g., “snort alert id”, “flow bytecount”, etc.)

# Network Data Model (cont'd)

- Develop *software sniffers*
  - understand native format of each device and translate to our common data model
  - support so far for PCAP, DAG, NetFlow, sFlow, 802.11 w/radio, any CoMo monitoring plug-in.
- Sniffers describe the packet stream they generate
  - Provide multiple templates if possible
  - Describe the fields in the schema that are available
  - Plug-ins just have to describe what they are interested in and the system finds the most appropriate matching

# Network Data Model (cont'd)

- Example: Cisco NetFlow sniffer
  - Regenerate packet stream from flow data
  - Augment packets with routing information (such as AS number, network prefix, etc)
  - Meta description will tell that 5-tuple information is there plus averaged packet sizes and timestamps (with accuracy equal to flow activity timer)
  - If re-processed, obtain same flow records



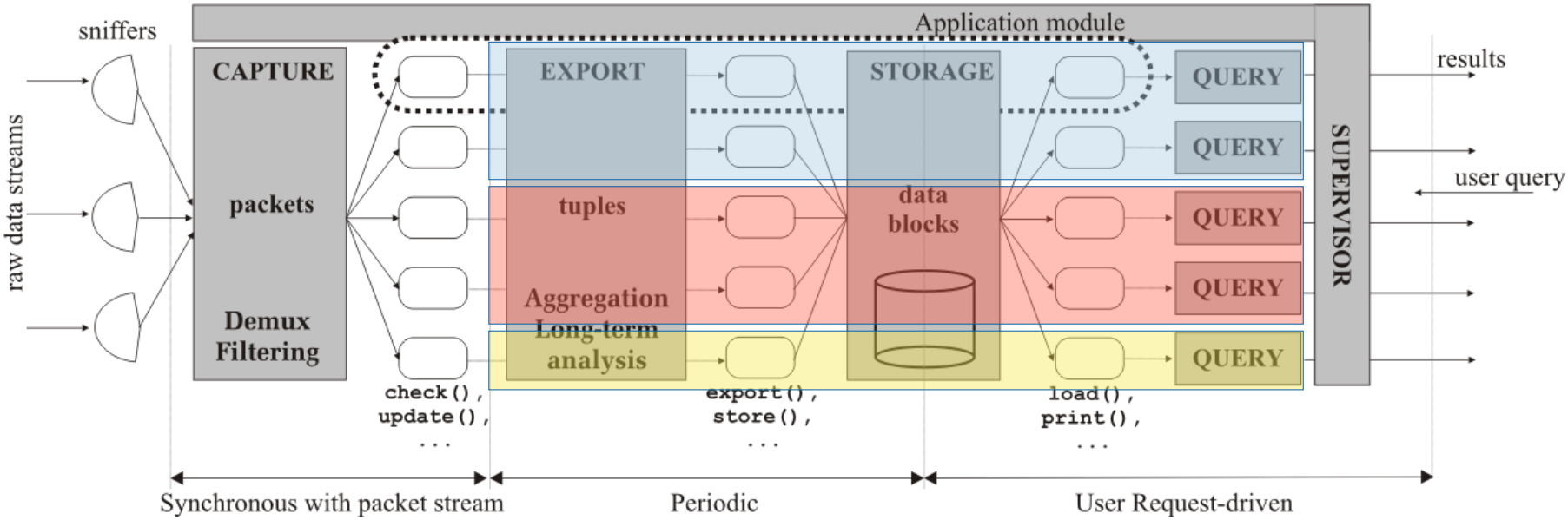
# Programming Model

- Application modules made of two components:
  *<filter>:<monitoring function>*
- Filter run by the core, monitoring function contained in the plug-in written by the user
  - set of pre-defined callbacks to perform simple primitives
  - e.g., update(), export(), store(), load(), print(), replay()
  - each callback is a closure (i.e., the entire state is defined in the call) so that it can optimized in isolation and executed anywhere.
- No explicit knowledge of the source of the packet stream
  - Modules specify what it needs in the stream and access fields via standard macros
  - e.g., IP(src), RADIO(snr), NF(src_as)

# Hardware Abstraction

- Goals: scalability and distributed queries
  - support large number of data sources and high data rates
  - support a heterogeneous environment (clients, APs, packet sniffers, etc.)
  - allow applications to perform partial query computations in remote locations
- To achieve this we...
  - hide to modules where they are running
  - enforce a programming structure
  - ... basically try to partially re-introduce declarative queries

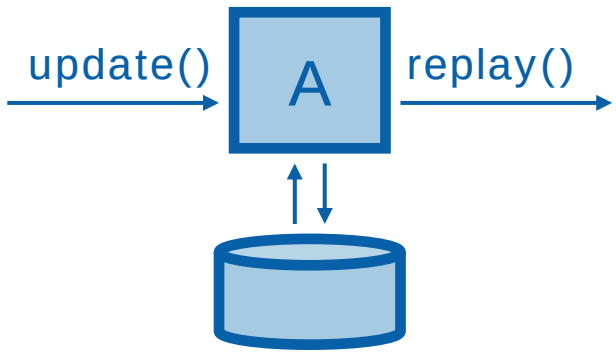
# Hardware Abstraction (cont'd)




- EXPORT/STORAGE can be replicated for load balancing
- CAPTURE is the main choke point
  - It periodically discards all state to reduce overhead and maintain a relative stable operating point

# Distributed queries

- Modules behave as software sniffers themselves
  - replay() callback to generate a packet stream out of module stored data
  - e.g., snort module generates stream of packets labeled with the rule they match; module B computes correlation of alerts
- This way computations can be distributed but also modules can be pipelined (to reduce the load on CAPTURE)

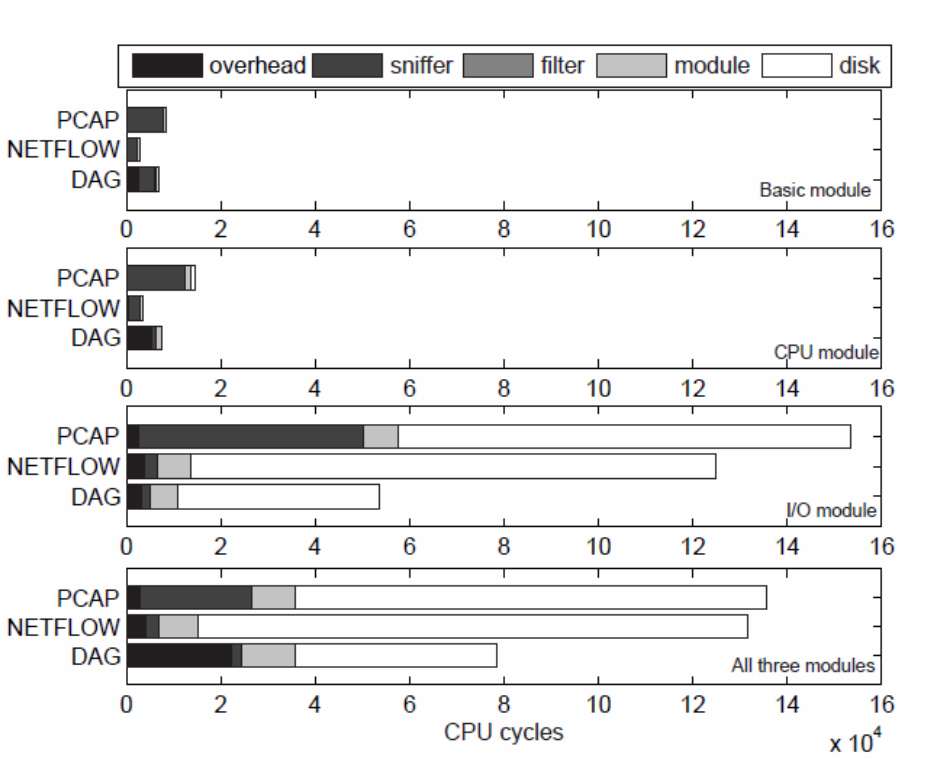


# Implementation

- Open source implementation
  - running on Linux, FreeBSD, Windows (w/Cygwin)
  - running on x86 and ARM architectures
  - supports PCAP, DAG, Netflow, sFlow, 802.11 w/radio
- Small set of application modules developed
  - Snort-like module for intrusion detection
  - Kismet-like module to detect wireless networks
  - Classical traffic statistics modules
- Support for continuous queries and triggers
  - Queries in the form “http://host:port/?module=...”
  - Developed graphical interface for queries (modules may send a gnuplot script with the print() callback)

# Early experiences

- Modules are rather simple to write and configure
  - Kismet → 127 C ";"
- Code base is robust. Current deployments:
  - Running over a GigE link with 700 Mbps avg. traffic
  - Running with over 180 modules concurrently
  - Running on Stargates using Compact Flash for storage without any change in the modules' code



# Related Work

- Gigascope [Cranor et al., Sigmod 2003]
  - GSQL to describe traffic query and schema. Possible to automatically offload to hardware some functions.
- FLAME [Anagnostakis et al., IWAN 2002]
  - Focus on safety and trust of in-kernel modules for network monitoring
- Aurora [Carney et al., VLDB 2002]
  - Handle (distributed) continuous queries on data streams. Seven operators and automated load shedding techniques
- Pandora [Patarin et al., Usenix 2000]
  - Construct dependency graph between individual monitoring components to perform a complex monitoring function
- Scriptroute [Spring et al., Usenix 2003]
  - Focus on making active measurement simpler to specify and run safely on a distributed architecture

# Conclusions and future work

- CoMo: an open platform for fast prototyping network measurement methods
- On-going and future work include
  - Enrich API adding more libraries and sniffers
  - Improve performance and add support for active storage
  - Support for GSQL or TelegraphCQ
  - Add static analysis of modules' code safety
  - Load Shedding
    - Use short-term resource usage prediction models to graceful degrade performance in presence of traffic anomalies

More info and source code at
**http://como.intel-research.net**

or send your questions to
**como-users@lists.sourceforge.net**