The CoMo project: An overview

Motivation

- Prototyping new traffic analysis methods is hard!
  - System performance strongly depends on traffic patterns
  - Long learning phase to fine tune heuristic parameters
  - Difficult to properly dimension system without prior knowledge of traffic characteristics

Motivation (cont'd)

- Need for an open network monitoring infrastructure
  - Widely deployed, diverse datasets
  - Supports multiple independent users
  - Provides a way to quickly implement the analysis methods
  - Abstracts away the internals of the network monitor

CoMo design goals

- Support multiple arbitrary traffic queries on the packet stream
- Arbitrary means that queries:
  - perform any computation on the packet stream
  - run on past traffic data as well as real-time
  - are coming from different independent users
- Multiple means that queries:
  - compete for resources on the entire infrastructure
  - may be scheduled to run on several systems at once

Related work

- Gigascope (AT&T)
  - GSQL to describe traffic query and schema. Possible to automatically offload to hardware some functions.
- FLAME (UPenn)
  - Focus on safety and trust of in-kernel modules for network monitoring
- Aurora, Borealis (MIT, Brown University)
  - Handle (distributed) continuous queries on data streams
  - Seven operators and automated load shedding techniques
- Scriptroute (UW)
  - Focus on making active measurement simpler to specify and run safely on a distributed architecture

CoMo System model

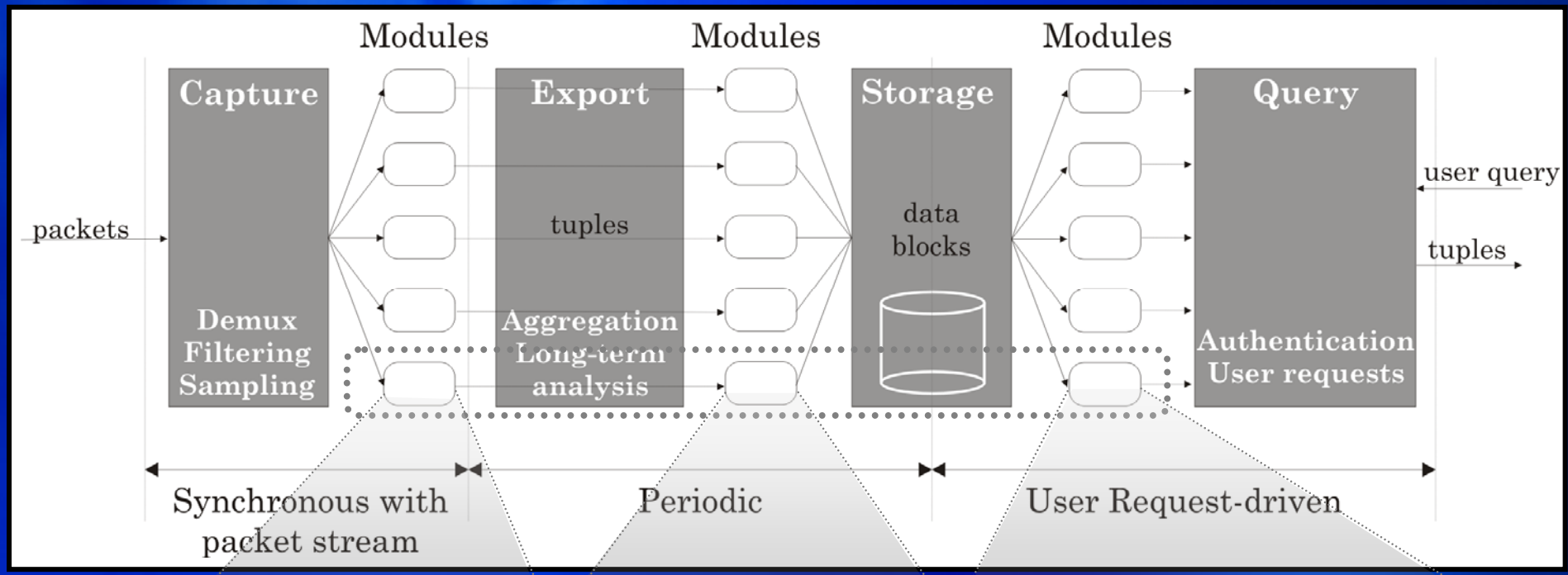
- Core processes
  - Handle all data movement (wire → memory → disk)
  - Unified interface to packet streams (abstracting capture devices)
  - Schedule queries and optimize allocation of computing resources
  - Maintain full packet trace for a window of time (e.g., 72 hours)
  - Provide query interface and handle user requests to access results

intel

System model (cont'd)

- Queries reside in plug-in modules
  - Defined by pair <filter:function>
  - Filter executed by core processes
  - Function to be applied on the packet stream specified in set of callbacks
    - Callbacks used to access query results too
    - Callbacks are closures (i.e. all state is defined in the call) to allow optimization by core processes
  - All state is maintained by core processes
    - Allows to stop/pause/resume modules when processing resources are scarce

System architecture



match();
update();
real-time computations on incoming packet stream

export();
store();
long-term analysis and storage to disk

load();
print();
retrieve data from disk (w/filtering and indexing) and format response

Writing query modules

- Success of system depends on how simple it is to write modules
  - Kismet module (114 semicolons)
  - Top-N destinations (62 semicolons)
  - Bytes/Packets Counter (53 semicolons)
- Each module maintains its own database
  - Using store(), load(), print() callbacks
  - Storage manager deals with indexing data and providing fast disk access.
  - Database schema known only to the module

intel

Historical queries

- Re-use computations made by modules
  - (only alternative is to go to packet trace)
- Modules provide a replay() callback
  - Runs on the module's database and generates a synthetic packet stream
  - Comes with a description of the stream
- Core system provides the best match
  - Find modules with all needed information
  - Choose based on storage size and processing costs of replay()

Historical queries (cont'd)

- Replay() allows significant reduction in disk reads at the cost of CPU cycles
  - Assumption is that CPU is not scarce while bandwidth from disk to memory is
- No explicit knowledge of what the two modules are actually doing!
- This method is used today to run on Dante's NetFlow data.
  - No need to modify queries that operate on live packet stream

intel

Hardware offload

- Off-loading some module callbacks to hardware.
- Easy for the filter, hard for callbacks
- Examples
  - load() code sent to active storage systems (e.g., DIAMOND) to search the database when no index exists
  - If update() is simple enough it can be compiled to run directly on an IXP's microengine.

Robustness

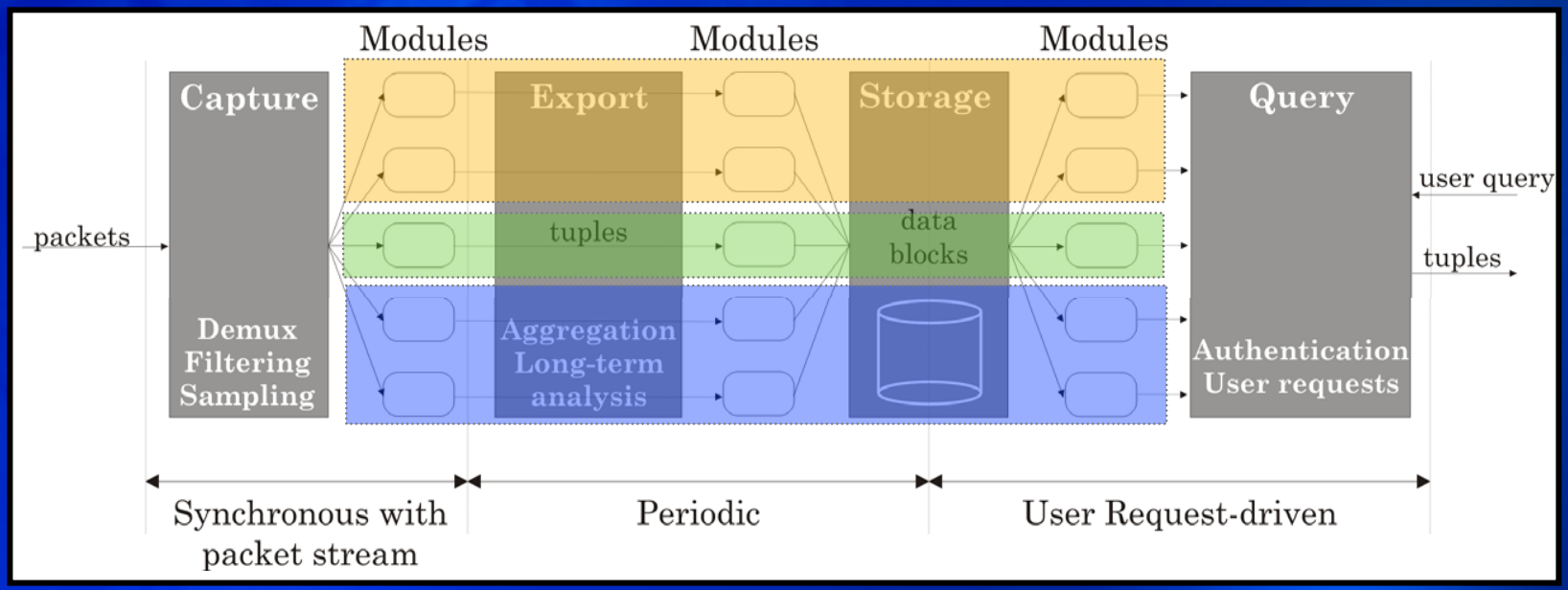
- Degrade module performance in presence of traffic anomalies or high query load
  - Module chosen based on resource usage
- Gracefully as module resource usage increases
  - Delay results (modules runs offline)
  - Sample input packet stream
  - Last resort: stop the module and resume it later
- Incentives for users to
  - Provide efficient implementations
  - Ask as few resources as possible
  - Implement replay() callback
    - if computation is reusable, it has higher priority

Achieving Safety

- Module may use disproportionate amount of resources
  - Same problem with legit modules and anomalous incoming traffic
- Module may corrupt other module's data
  - Removing pointer arithmetic would solve the problem but it's not feasible
  - Use approach similar to FLAME (based on Cyclone)
- Module may break usage policy
  - e.g., break anonymization scheme
  - Local anonymization is easy to do but not good enough for most users
  - Distributed anonymization still an open problem…

Achieving Scalability

- Running on cluster of nodes
  - "similar" modules run on same node



Making it distributed…

- Demands for global resource management
- Online queries: Optimal Network-wide Sampling
  - Select the nodes that need to run a specific module to minimize the total number of packet processed
  - Module's utility function that depends on the aggregate sampling rate across the network of monitors
  - Optimal algorithm for a class of utility functions
- Retrospective queries: MIND distributed index
  - Nodes share and distribute some information on the traffic they observe
  - Query first look in the index to find nodes that may contain data of interest
  - Prototype implemented on PlanetLab

Conclusion

- CoMo is a platform for fast prototyping traffic analysis methods
- Implementation publicly available
  - Runs on x86 and StrongARM;
  - Supports many network capture devices (DAG, libpcap, Prism2 wireless cards, NetFlow)
  - Full packet capture on Gigabit links
  - Four code releases so far…
- Looking for users and sites where to develop new nodes

intel

More info

Source code available at
http://como.sourceforge.net

como-users@lists.sourceforge.net
for comments, questions, etc.