Building CoMo

Pere Barlet Ros
Joint work with: Euan Harris, Gianluca Iannaccone and Christophe Diot

Intel Research Cambridge
September 10th, 2004
Outline

- What is CoMo?
- Challenges
- Architecture
- Modules
- Queries
- Current status
- Future work
What is CoMo?

• Large community working on monitoring
  – Uncoordinated efforts, different datasets, difficult to validate other people’s research, etc.

• CoMo as a building block of an open infrastructure
  – High speed always-on network monitoring system
  – Flexible, scalable and run on commodity hardware
  – Users can define their own metrics and queries
Challenges

• Ease of deployment
• Querying the network data
• Resource Management
• Security, Privacy and Confidentiality
Addressed challenges

• Ease of deployment
  – Implement new methods to analyse data traffic
  – Availability of diverse datasets

• Querying the network data
  – Express and run queries without explicit built-in support
  – Predict and reduce query response time

• Resource Management

• Security, Privacy and Confidentiality
CoMo Architecture

- Core engine provides basic primitives
  - an interface to plug-in the module
  - an interface to access the data computed by the modules
- Traffic metrics reside in plug-in modules
  - Compute one specific metric on the traffic data
  - Pre-compute queries
  - Activated on demand
  - Users can “push” module to the infrastructure
Data flow view

- **Strict decoupling between processes**
  - Capture & Storage meant to isolate processes
  - Modules organized as set of callbacks
  - Each process could run on different hardware (e.g., Capture on IXP platform)
Module design and requirements

• Modules are independent of each other
  – No state information passed between modules
  – Implemented independently by users

• Modules are very restricted on what they can do
  – Limited access to system calls, memory, etc.
  – Complex functions are implemented by core
Module details

- Modules are defined by the pair
  - `<filter>`:`<function>`
- Filter is compiled on-the-fly by CoMo
  - E.g. ‘proto(TCP) && TCP(dst_port) == 80’
  - Run within CAPTURE
- Function is built as a shared object
  - Dynamically linked into CoMo
  - Standardised entry points
  - Easy to write
    - E.g. utilization: 20 ;’s
    - E.g. top destinations: 16 ;’s
What does a module look like?

- Modules provide set of pre-defined callbacks
- One set for packet processing
  - to process individual packets
  - used by CAPTURE
- One set for long-term analysis
  - to process state information
  - used by EXPORT
- One set to access data on disk
  - to store and load information to/from disk
  - used by EXPORT and QUERY
# Packet-processing callbacks

<table>
<thead>
<tr>
<th>Callback</th>
<th>Called by</th>
<th>Function</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>init()</td>
<td>supervisor</td>
<td>init additional module data structures and arguments</td>
<td>do nothing</td>
</tr>
<tr>
<td>check()</td>
<td>capture</td>
<td>check if a packet is acceptable for the module</td>
<td>always accept</td>
</tr>
<tr>
<td>hash()</td>
<td>capture</td>
<td>compute a hash key a capture record</td>
<td>return 0</td>
</tr>
<tr>
<td>match()</td>
<td>capture</td>
<td>check if a capture record is already in the hash table</td>
<td>always match</td>
</tr>
<tr>
<td>new()</td>
<td>capture</td>
<td>return the size of a capture record</td>
<td>default size</td>
</tr>
<tr>
<td>update()</td>
<td>capture</td>
<td>update the capture record with data of a packet</td>
<td>---</td>
</tr>
</tbody>
</table>
## Long-term analysis callbacks

<table>
<thead>
<tr>
<th>Callback</th>
<th>Called by</th>
<th>Function</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep()</td>
<td>export</td>
<td>indicate if export data has to be kept</td>
<td>always keep</td>
</tr>
<tr>
<td>ehash()</td>
<td>export</td>
<td>compute hash key for an export record</td>
<td>same key as hash()</td>
</tr>
<tr>
<td>ematch()</td>
<td>export</td>
<td>check if an export record is already in hash table</td>
<td>always match</td>
</tr>
<tr>
<td>enew()</td>
<td>export</td>
<td>return the size of an export record</td>
<td>default size</td>
</tr>
<tr>
<td>eupdate()</td>
<td>export</td>
<td>update an export record with data of a capture record</td>
<td>---</td>
</tr>
<tr>
<td>ecompare()</td>
<td>export</td>
<td>compare two export records to sort them</td>
<td>FIFO</td>
</tr>
<tr>
<td>discard()</td>
<td>export</td>
<td>indicate if an export record has to be expired or stored</td>
<td>no expire and store</td>
</tr>
</tbody>
</table>
## File access callbacks

<table>
<thead>
<tr>
<th>Callback</th>
<th>Called by</th>
<th>Function</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>store()</td>
<td>export</td>
<td>return which data of a capture record must be stored to disk</td>
<td>no store</td>
</tr>
<tr>
<td>estore()</td>
<td>export</td>
<td>return which data of a export record must be stored to disk</td>
<td>no store</td>
</tr>
<tr>
<td>load()</td>
<td>query</td>
<td>given a chunk of a file red from disk return the size of the next capture record and its timestamp</td>
<td>no load</td>
</tr>
<tr>
<td>eload()</td>
<td>query</td>
<td>given a chunk of a file red from disk return the size of the next export record and its timestamp</td>
<td>no load</td>
</tr>
</tbody>
</table>
Configuration example

...  
module 'top-dst-irc'
  description 'Most common destinations from IRC users'
  weight 100
  filter 'outgoing && proto(TCP) && srcip(10.212.4.0, 24)'
  library 'ranking'
  output 'top.10.dstip.irc'
  args '--field=dst-ip --top=10'
  window 30m
  period 72h
end

module 'top-src-irc'
  description 'Most common source to IRC network'
  weight 50
  filter 'incoming && proto(TCP) && dstip(10.212.4.0, 24)'
  library 'ranking'
  output 'top.10.srcip.to.irc'
  args '--field=src-ip --top=10'
  window 30m
  period 72h
end

library 'ranking'
  source 'ranking.so'
  hashsize 64K
end

...
What does a query look like?

- CoMo maintains a large amount of data
  - Modules pre-compute as much as possible
  - Random query needs to find the module that is pre-computing the data
- Static query: `sendto <ip>:<port>`
  - it knows the module by configuration
- On-demand query: carries the module
  - query goes with source code
  - if no module matches, run the code or look at packet trace
- Ad-hoc query: module compiled on the fly
  - query written in specific language (SQL-like)
Example of a static query

- A collector receives pre-computed data from a module for static-queries
- Output can be redirected to a file, post-processing programs, graph generators, etc.
- `como-collector -f <format> -p <port>`
  - `-f`: format as in printf (with additional features)
  - `-p`: port to listen for query results
Example of an on-demand query

• A client sends on-demand queries to a CoMo box and receives the query results
• como-client -f <format> -i <address> -p <port> -t <filter> -c <classifier> -s <start> -e <end> -x
  – -f: format as in printf
  – -i, -p: IP address/port to send the query
  – -t: filter to be applied on the traffic stream
  – -c: name of the classifier (function)
  – -s, -e: interval query is interested on
  – -x: query export data (otherwise capture data)
Current status

- Sniffers
  - Live capture: BPF, DAG
  - Traces: pcap, ERF
- Capture
- Export
- Static and part of on-demand queries
- A basic set of modules
  - packet trace, packet counter, link utilization, application breakdown, protocol breakdown, flow classification, top destinations
Current status (cont’d)

- Running at IRC

![Application breakdown (bits/s)](image1)

![Application breakdown (pkts/s)](image2)
Future work

- Complete on-demand query support
  - Load modules on-demand
  - Run modules in packet trace
  - Reuse information pre-computed by other modules

- Resource management
  - The system should control its resources carefully
    - Traffic characteristics
    - Modules
    - Queries