Composable IPFIX Mediation with BlockMon

Brian Trammell <trammell@tik.ee.ethz.ch>
Communication Systems Group, ETH Zürich
BlockMon: Concepts and Goals

- Composable measurement using small blocks
  - Increases measurement performance on multicore hardware
  - Code reuse for measurement development
- Develop platform for understanding composable measurement application development
- Enable code and analysis interchange in the form of compositions of modules from a standard, trusted base
- Support multi-stage streaming analysis through the use of mediators and designed-in support for IPFIX.
- *This is all very much a work in progress...*
BlockMon

- Compositions of *blocks* send *messages* connected via *gates*.
- Framework, scheduling, and blocks in C++11 (gcc-4.5+)
- Python-based CLI and JSON-RPC daemon support configuration through XML-based composition schema
- Developed by NEC, CNIT, and ETH within the FP7-DEMONS project, available under BSD license
BlockMon performance

- Designed to leverage modern multicore hardware
- Rotating queues minimize lock contention
- Batching reduces allocation overhead
- Immutable messages with tagging for efficient pipelining
- Current work on making things even faster
  - cache-aware message allocation
  - queue instrumentation and automated tuning
- → When capturing packets, can handle 10Gbps at line rate on modern commodity multicore hardware
libfc

- BlockMon uses libfc, a new IPFIX implementation in C++
  - Design based on ripfix, presented at NMRG workshop in Maastricht
- Design for high-speed export
  - Direct transcoding from C structures into IPFIX Messages
- Design for high-speed, flexible collection
  - Polymorphism by minimal template matching
  - Direct transcoding from IPFIX Messages into C structures
  - Collection by set (not by record) to allow buffering, etc.
- Future design support for bridging to generic record storage in STL containers
  - slow by design, very much not there yet…
Minimal template matching

- Support for collection of multiple complex data types (templates) required for record to message mapping
- Each record type represented by a *minimal template*
  - Information Elements which must be present to consider it a template representing that type.
  - Conflict resolution by list minimal template list order.
- Templates matched to record type by minimal template match, cached by SetID
- → Open issue: not very structured data (RFC6313) friendly
BlockMon and IPFIX

- Msg base class supports bridging between IPFIX Records and BlockMon Messages
  - Each message can represent itself as a raw C struct for transcoding.
  - Each message class has a Descriptor.
  - Descriptor can return IPFIX templates for wire and struct encoding for a given message class.
  - Descriptor can create a new message object given a raw C struct.
- IPFIXExporter/IPFIXSource blocks implement EP and CP.
BlockMon as Mediator

- Any composition of blocks between an IPFIXSource and an IPFIXExporter is a Mediator as in RFC 6183.
- Built-in Flow message type models a uniflow and supports basic flow mediation use cases
  - flow{Start,End}Milliseconds
  - {octet,packet}DeltaCount
  - {source,destination}IPv4Address
  - {source,destination}TransportPort
  - protocolIdentifier
Use cases

- Simple mediators as in RFC 6183
  - Simple property-based flow selection: FlowFilter block allows parametric selection based on flow key
  - Simple IP anonymization as in RFC 6235: IPAnonymizer rewrites IP addresses in Flows
- Application-specific aggregation and correlation
- Analysis offload in cluster environments
  - IPFIX allows a composition to be split across nodes
Use case: simple flow filtering

```
<block id="collector" type="IPFIXSource" invocation="async" threadpool="collect_thread">
  <params>
    <collect transport="udp"/>
  </params>
</block>

<block id="filter" type="FlowFilter" invocation="direct">
  <params>
    <filter_mode behavior="discard"/>
    <l4_protocol number="6">
      <filter_mode behavior="accept"/>
    </l4_protocol>
    <dst_port number="80">
      <filter_mode behavior="accept"/>
    </dst_port>
  </params>
</block>

<block id="filter" type="IPFIXExporter" invocation="indirect" threadpool="exportt_thread">
  <params>
    <export transport="udp" host="1.2.3.4"/>
  </params>
</block>
```
Use case: simple flow filtering

```
<block id="collector" type="IPFIXSource" invocation="async" threadpool="collect_thread">
  <params>
    <collect transport="udp"/>
  </params>
</block>

<block id="filter" type="FlowFilter" invocation="direct">
  <params>
    <filter_mode behavior="discard"/>
    <l4_protocol number="6"><filter_mode behavior="accept"/></l4_protocol>
    <dst_port number="80"><filter_mode behavior="accept"/></dst_port>
  </params>
</block>

<block id="filter" type="IPFIXExporter" invocation="indirect" threadpool="exportt_thread">
  <params>
    <export transport="udp" host="1.2.3.4"/>
  </params>
</block>
```
Use case: simple flow filtering

```
<block id="collector" type="IPFIXSource" invocation="async" threadpool="collect_thread">
  <params><collect transport="udp"/></params>
</block>

<block id="filter" type="FlowFilter" invocation="direct">
  <params>
    <filter_mode behavior="discard"/>
    <l4_protocol number="6"><filter_mode behavior="accept"/></l4_protocol>
    <dst_port number="80"><filter_mode behavior="accept"/></dst_port>
  </params>
</block>

<block id="filter" type="IPFIXExporter" invocation="indirect" threadpool="exportt_thread">
  <params><export transport="udp" host="1.2.3.4"/></params>
</block>
```
Use case: simple flow filtering

![Diagram of flow filtering process]

```
<block id="collector" type="IPFIXSource" invocation="async" threadpool="collect_thread">
  <params><collect transport="udp"/></params>
</block>

<block id="filter" type="FlowFilter" invocation="direct">
  <params>
    <filter_mode behavior="discard"/>
    <l4_protocol number="6"><filter_mode behavior="accept"/></l4_protocol>
    <dst_port number="80"><filter_mode behavior="accept"/></dst_port>
  </params>
</block>

<block id="filter" type="IPFIXExporter" invocation="indirect" threadpool="export_thread">
  <params><export transport="udp" host="1.2.3.4"/></params>
</block>
```
Use case: simple flow filtering

```
<connection src_block="collector" src_gate="source_out"
           dst_block="filter" dst_gate="in_flow"/>

<connection src_block="filter" src_gate="out_flow"
           dst_block="exporter" dst_gate="in_rec"/>
```
The bad news

- BlockMon under very active development
  - Currently available open-source release is really old
  - New BSD-licensed open-source release planned “soon”
  - Code available upon request in the meantime
  - Internal APIs changing as performance research continues
- IPFIX support in BlockMon alpha quality
  - libfc will be released separately as soon as it’s stable
- trammell@tik.ee.ethz.ch
  - if you want to play with any of this
Acknowledgments

- FP7-DEMONS project
- The BlockMon team
  - NEC Laboratories Europe
  - CNIT (Consorzio Nationale Interuniversitario per le Telecomunicazioni)
    - University of Pisa
    - University of Rome Tor Vergata