IBM

Zurich Research Laboratory

Simplifying the configuration of flow monitoring probes

Xenofontas (Fontas) Dimitropoulos (xed@zurich.ibm.com)
Andreas Kind (ank@zurich.ibm.com)

Outline

- Background and motivation.
- Probe configuration architecture:
  - Requirements and goals.
  - Design.
  - Implementation.
- Future work and conclusions.

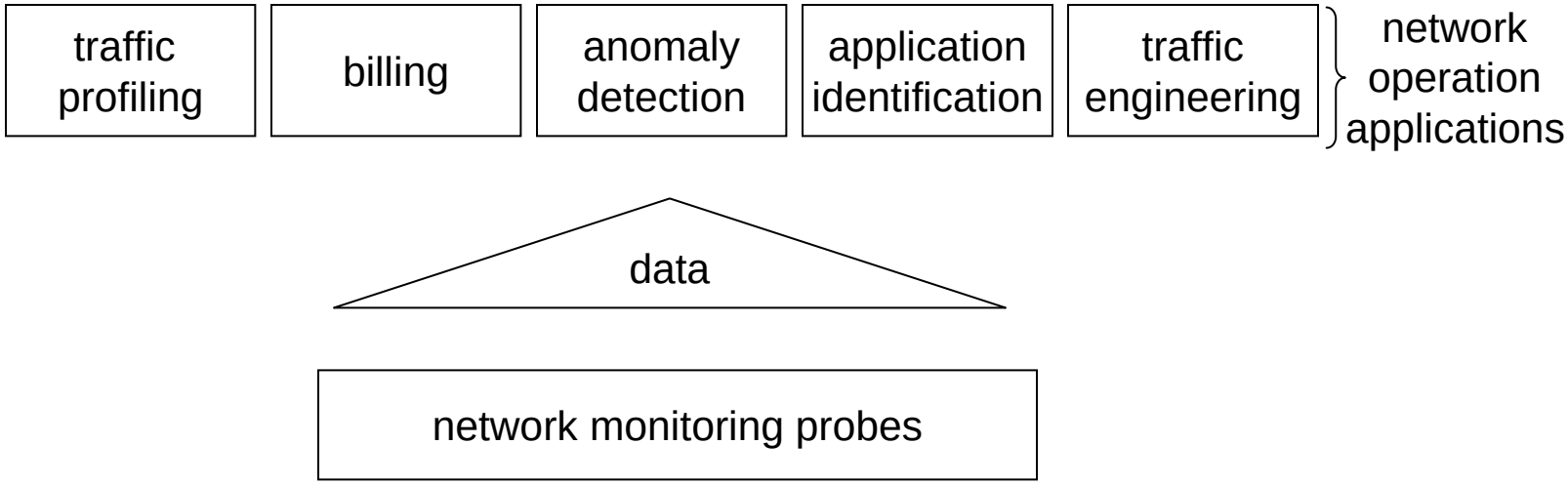
Network configuration

- Network elements are typically configured with low-level commands, e.g., Cisco IOS commands.
- Network administrators manage numerous network elements with lengthy configuration files.
- Network configuration is an error-prone and time-consuming process.
- Configuration errors can be costly, e.g.:
  - network outages
  - violations of SLAs

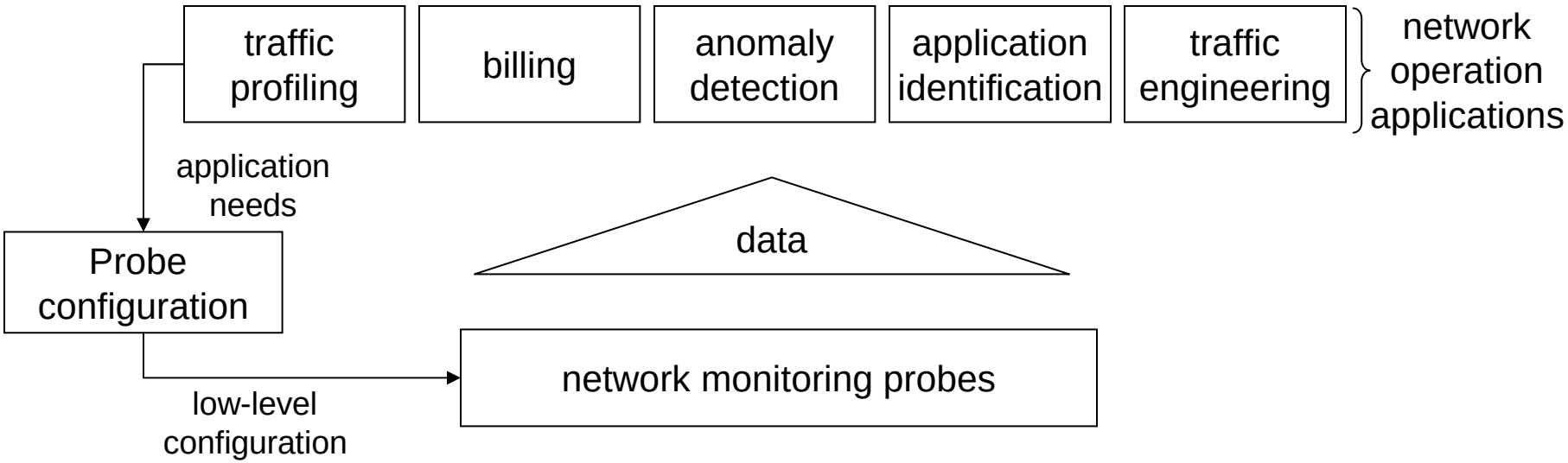
Probe configuration

- The configuration of monitoring probes is part of the more general network configuration problem.
- Monitoring probes are gradually becoming more intelligent, for example, using advanced sampling and data aggregation techniques. Consequently, their configuration becomes more involved.
- Flexible Netflow (FNF) and IPFIX provide numerous configuration options that were not available earlier:
  - FNF has 58 different configuration commands.
  - FNF provides 65 different fields, arbitrary combinations of which can be used in the definition of flow key and non-key fields.
- Certain network operation applications need to dynamically change configuration to:
  - adapt to changing traffic conditions.
  - investigate on-going network anomalies.

Configuration requirements



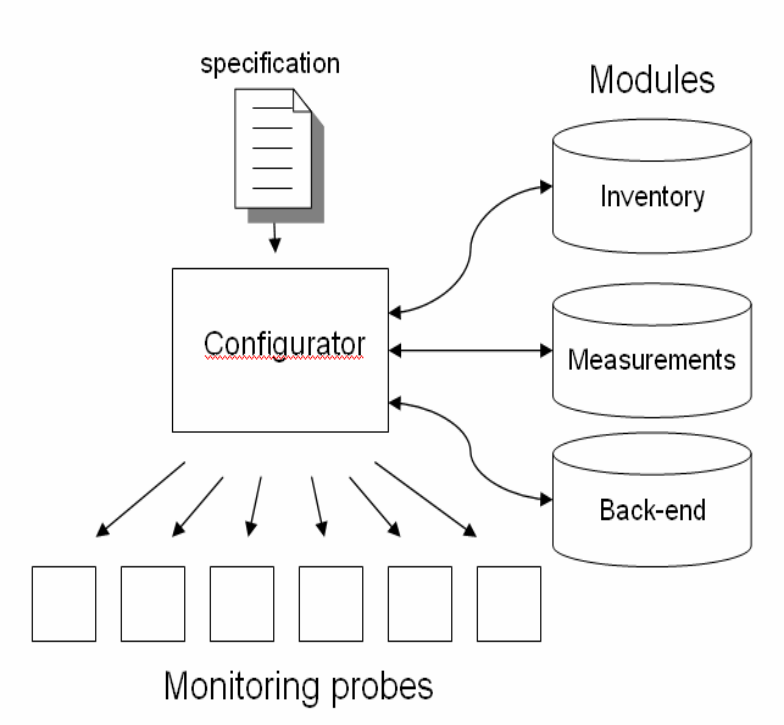
Configuration requirements



- Probe configuration should:
  1. take into account application needs.
  2. be aware of the available monitoring probes.
  3. generate low-level configuration commands.
  4. configure or update the configuration of probes.

Probe configuration architecture

- Three modules:
  - the measurements module describes different measurements, i.e., application needs.
  - the inventory module describes the monitoring probes of a network.
  - the back-end module provides necessary information for generating low-level commands.
- The specification identifies application needs.
- The configurator:
  - uses the modules and specification to generate low-level commands.
  - configures the probes

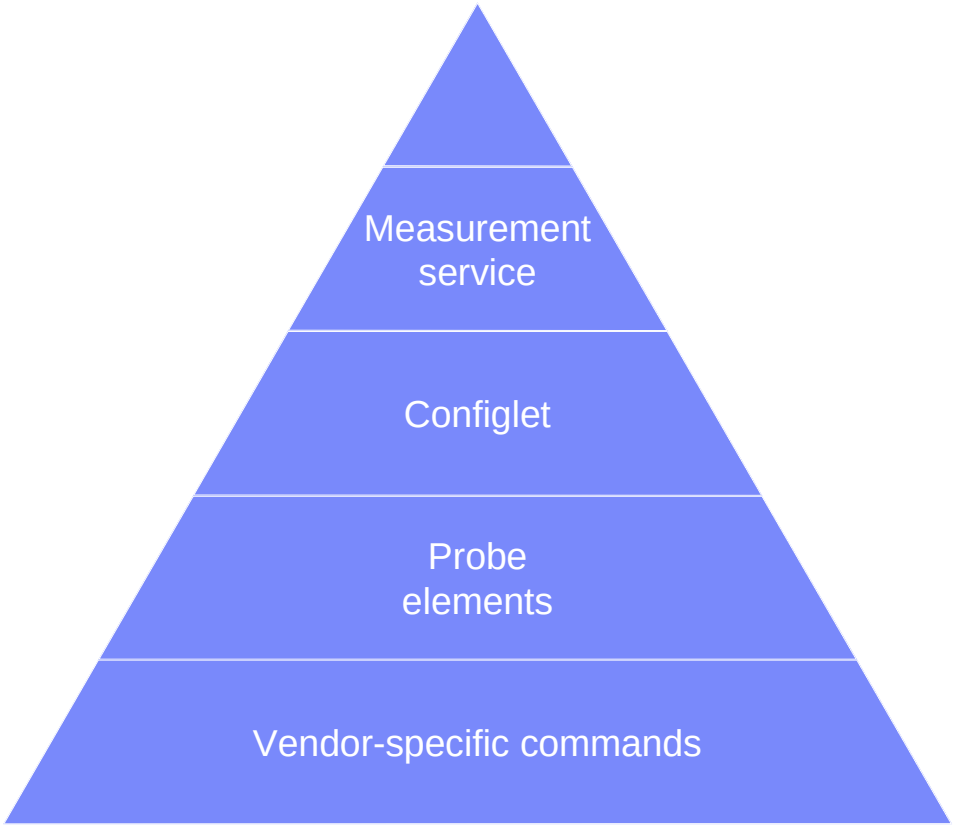


Design goals for simplifying configuration

1. Abstraction: hide low-level configuration commands.
2. Objective-oriented configuration expression:
   - express configuration in terms of measurement objectives.
   - focus on measurements instead of devices.
3. Network-wide configuration: configure a network instead of configuring individual devices.
4. Re-usability: make parts of configuration network-independent.
5. Extensibility: easily introduce support for new commands, measurements, etc.

Configuration abstraction hierarchy

- 1st level: vendor-specific configuration commands.
- 2nd level: probe elements (pe), i.e., logical components of a probe, like interface, flow cache, exporter.
- 3rd level: configlet, i.e., a set of specific probe elements that realizes a measurement.
- 4th level: measurement services, i.e., a configlet with certain probe selection rules.



Back-end module

- Specifies different probe elements.
- A probe element specification:
  - is written in XML.
  - has a unique id.
  - identifies parameters and parameter default values.
  - determines the low-level vendor-specific commands.

```
<!– Probe Element Exporter -->
<pe id='generic_exporter'>

 <params>
  <param id='port'>90</param>
  <param id='transport'>udp</param>
  <param id='destination'>192.0.0.1</param>
  <param id='label'>EXPORTER</param>
 </params>

 <template>
  <ios>
        flow exporter $label
        destination $destination
        transport $transport $port
  </ios>
  <yaf>
        --out $destination --ipfix $transport --ipfix-port $port
  </yaf>
  <junos>
  </junos>
 </template>

</pe>
```

Inventory module

- Specifies network probes, i.e., lists the characteristics that can be useful for their configuration.
- Besides describing location, system, and interface information, it declares tags that can be used for grouping probes and for probe selection.

```
<probe id='trabant.zurich.ibm.com'>
  <address>9.4.68.154</address>

  <location>
    <city>Zurich</city>
    <state>Central CH</state>
    <country>Switzerland</country>
  </location>

  <system>
    <os>ios</os>
    <version>12.4</version>
  </system>

  <interface id='FastEthernet0/0'>
    <capacity>100Mbits</capacity>
    <tag>internal</tag>
  </interface>

  <interface id='FastEthernet0/1'>
    <capacity>100Mbits</capacity>
    <tag>customer</tag>
  </interface>

  <tags>
    <tag>edge</tag>
  </tags>

 </probe>
```

Measurements module

```
<!--   Monitor how much traffic is send -->
<!--   between IP blocks. -->
<msr id='traffic_matrix'>

  <params> <!-- Default parameter values -->
    <param id='collector_address'>localhost</param>
    <param id='collector_port'>2055</param>
    <param id='collector_transport'>tcp</param>
  </params>

  <!-- Probe element chain -->
  <configlet>
  </configlet>

  <rules>
  </rules>

</msr>
```

Measurements module

```
<!-- Probe element chain -->
<configlet>
   <pe>
            <name>exporter</name>
            <params>
              <param id='label'>TM_EXPORTER</param>
              <param id='destination'>$collector_address</param>
              <param id='port'>$collector_port</param>
              <param id='transport'>$collector_transport</param>
            </params>
   </pe>
   <pe>
            <name>flow_cache</name>
            <params>
              <param id='label'>TM_CACHE</param>
              <param id='record'>SRC_DST_PREFIX_REC</param>
              <param id='export'>TM_EXPORTER</param>
            </params>
   </pe>
   <pe>
            <name>interface</name>
            <params>
              <param id='monitor'>TM_CACHE</param>
              <param id='interface'>$interface->id</param>
              <param id='direction'>output</param>
            </params>
   </pe>
</configlet>
```

Measurements module

```
<rules>
    <interface>
                if ( $interface.tag eq “external" and
                     $probe.tag eq "edge" ) {
                    return 1;
                } else {
                    return 0;
                }
    </interface>
</rules>
```

Input specification

- Lists the measurements and the probes in which to enable these measurements.
- Is the user interface and can be generated through a GUI.

```
<!-- Probes to apply measurements on -->
<probe id='wassen.zurich.ibm.com'></probe>
<probe id='trabant.zurich.ibm.com'></probe>


<!-- Measurements -->
<msr id='traffic_matrix'>
   <params> <!-- overwrite default values -->
     <param id='collector_address'>9.4.68.204</param>
     <param id='collector_port'>2055</param>
     <param id='collector_transport'>udp</param>
   </params>
</msr>

<msr id='app_monitoring'>
   <params> <!-- overwrite default values -->
     <param id='collector_address'>9.4.68.205</param>
     <param id='collector_port'>2055</param>
     <param id='collector_transport'>udp</param>
   </params>
</msr>
```

Design goals for simplifying configuration

1. Abstraction: hide low-level configuration commands.
2. Objective-oriented configuration expression:
   - express configuration in terms of measurement objectives.
   - focus on measurements instead of devices.
3. Network-wide configuration: configure a network instead of configuring individual devices.
4. Re-usability: make parts of configuration network-independent.
5. Extensibility: easily introduce support for new commands, measurements, etc.

Conclusions

- Described an architecture for automating the configuration of flow monitoring probes.
  - Configuration abstraction.
  - Reuse configuration.
  - Extensibility.
- Future/on-going work:
  - Incorporate error-checking techniques.
  - Develop libraries for typical measurements.
  - Configuration optimization.
  - Use NetConf.