Implementation of SCRIPT Mechanisms

Cristian Morariu

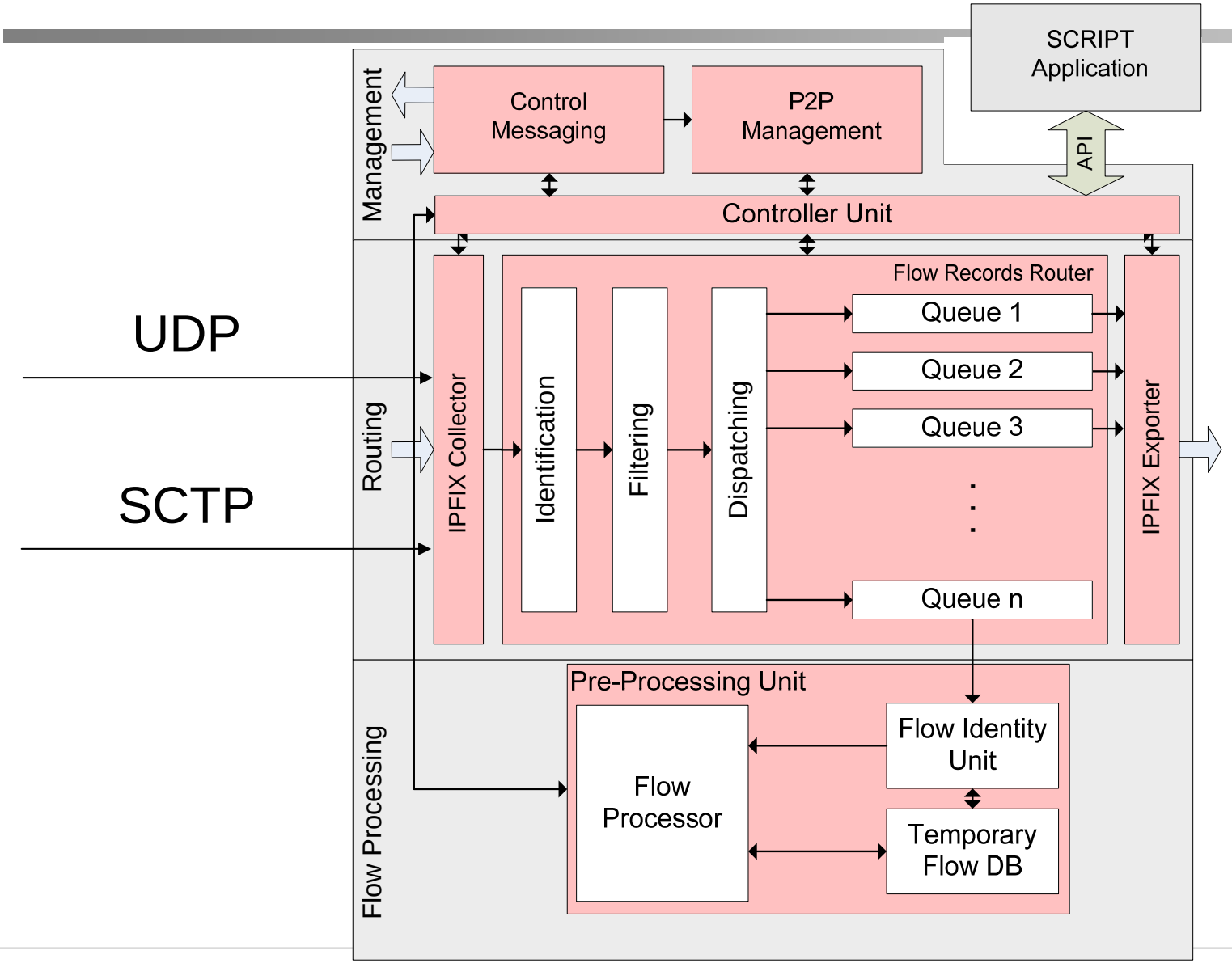
Communication Systems Group CSG
Department of Informatics IFI
University of Zürich UZH

ifi

Overview

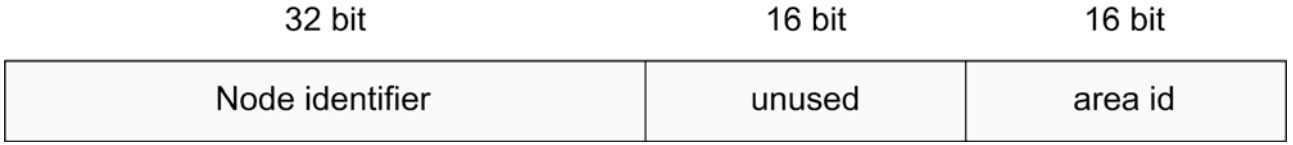
- SCRIPT Node Architecture
- Node identifiers and Routing Hash IDs
- Additional template fields
- Template synchronization
- Routing process
- Handling of flow records for unavailable nodes
- SCRIPT Monitoring Tool
- Steps to implement a SCRIPT application

SCRIPT Node Architecture

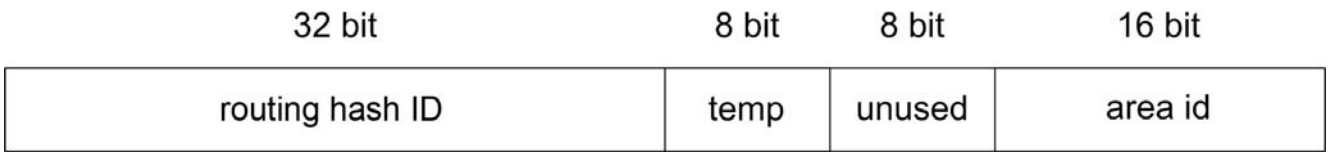


Node Identifier and Routing Hash ID

- Each SCRIPT node has a 64 bit identifier



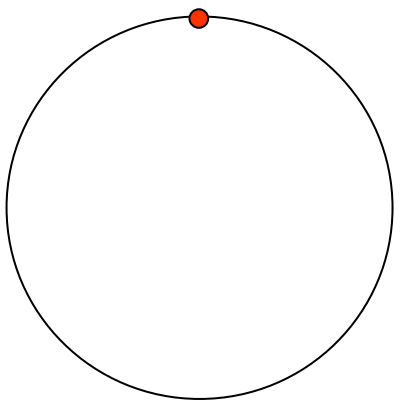
- Last 16 bit represent an area identifier
  - Used if localized storage of flow records is desired
    - *e.g. flow records from CH should only be stored by nodes in CH*
- For each flow record, a routing hash ID is calculated



- SCRIPT nodes will route each flow record so it reaches the SCRIPT node with the lowest ID which is larger than the routing hash ID.

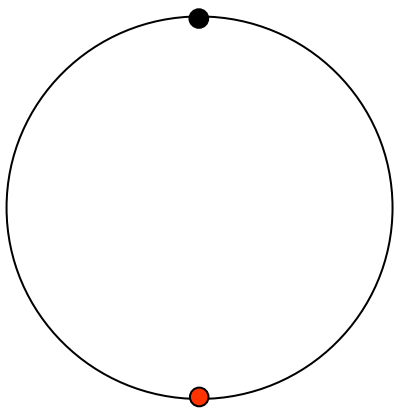
Identity Assignment

- When a SCRIPT node starts, it sends an identity request to the SCRIPT Controller
- SCRIPT Controller maintains a list of all existing nodes and their IDs
- It chooses the biggest empty interval in the identity space and calculates the middle of that interval
- The first node will also receive the identity 0xFFFFFFFF
- The new node will be assigned that identity
- *E.g. 7 nodes join:*
  - *Node 1: 0xFFFFFFFF*



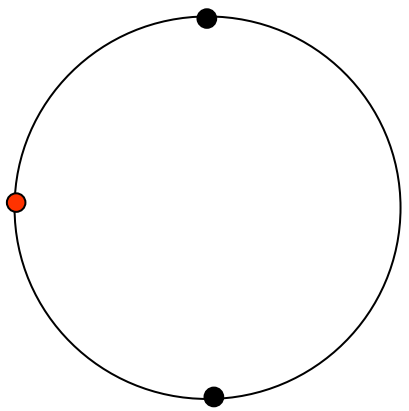
Identity Assignment

- When a SCRIPT node starts, it sends an identity request to the SCRIPT Controller
- SCRIPT Controller maintains a list of all existing nodes and their IDs
- It chooses the biggest empty interval in the identity space and calculates the middle of that interval
- The first node will also receive the identity 0xFFFFFFFF
- The new node will be assigned that identity
- *E.g. 5 nodes join:*
  - *Node 1: 0xFFFFFFFF*
  - *Node 2: 0x7FFFFFFF*



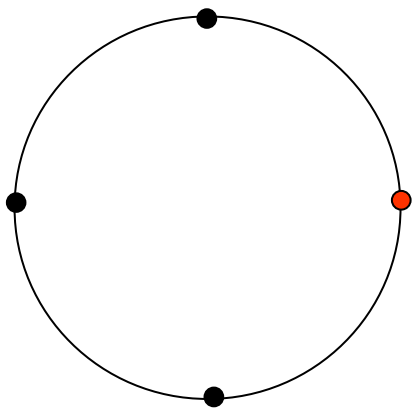
Identity Assignment

- When a SCRIPT node starts, it sends an identity request to the SCRIPT Controller
- SCRIPT Controller maintains a list of all existing nodes and their IDs
- It chooses the biggest empty interval in the identity space and calculates the middle of that interval
- The first node will also receive the identity 0xFFFFFFFF
- The new node will be assigned that identity
- *E.g. 7 nodes join:*
  - *Node 1: 0xFFFFFFFF*
  - *Node 2: 0x7FFFFFFF*
  - *Node 3: 0xBFFFFFFF*



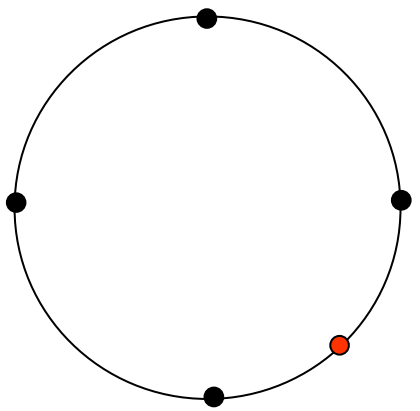
Identity Assignment

- When a SCRIPT node starts, it sends an identity request to the SCRIPT Controller
- SCRIPT Controller maintains a list of all existing nodes and their IDs
- It chooses the biggest empty interval in the identity space and calculates the middle of that interval
- The first node will also receive the identity 0xFFFFFFFF
- The new node will be assigned that identity
- *E.g. 7 nodes join:*
  - *Node 1: 0xFFFFFFFF*
  - *Node 2: 0x7FFFFFFF*
  - *Node 3: 0xBFFFFFFF*
  - *Node 4: 0x3FFFFFFF*



Identity Assignment

- When a SCRIPT node starts, it sends an identity request to the SCRIPT Controller
- SCRIPT Controller maintains a list of all existing nodes and their IDs
- It chooses the biggest empty interval in the identity space and calculates the middle of that interval
- The first node will also receive the identity 0xFFFFFFFF
- The new node will be assigned that identity
- *E.g. 7 nodes join:*
  - *Node 1: 0xFFFFFFFF*
  - *Node 2: 0x7FFFFFFF*
  - *Node 3: 0xBFFFFFFF*
  - *Node 4: 0x3FFFFFFF*
  - *Node 5: 0x5FFFFFFF*



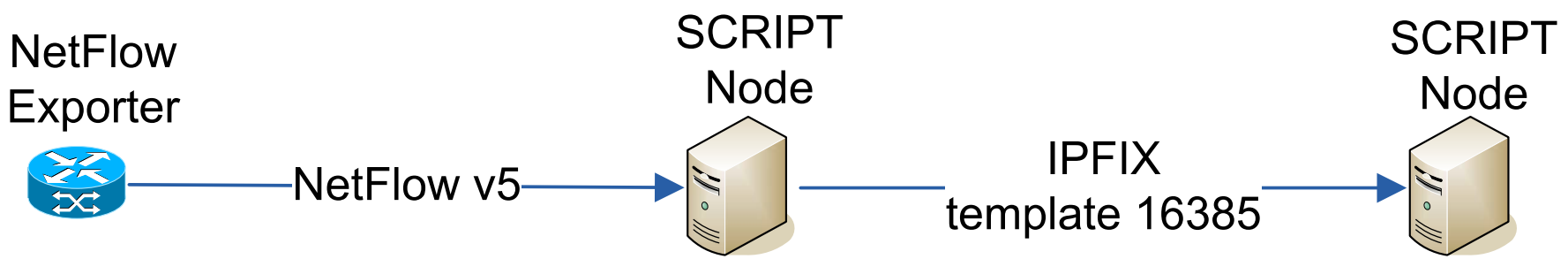
Extra Template Fields

- Single collector → exporter ID in NetFlow packet header
- SCRIPT → flow records in the same exported packet may be distributed, so NetFlow packet header info is lost
- Exporter ID is inserted in each flow record (4 byte)
- Routing Hash ID is also inserted in every record (8 byte)
  - Extra network overhead, but hash does not need to be calculated at every node
- Exporter sysUpTime also included as an extra field (4 byte)

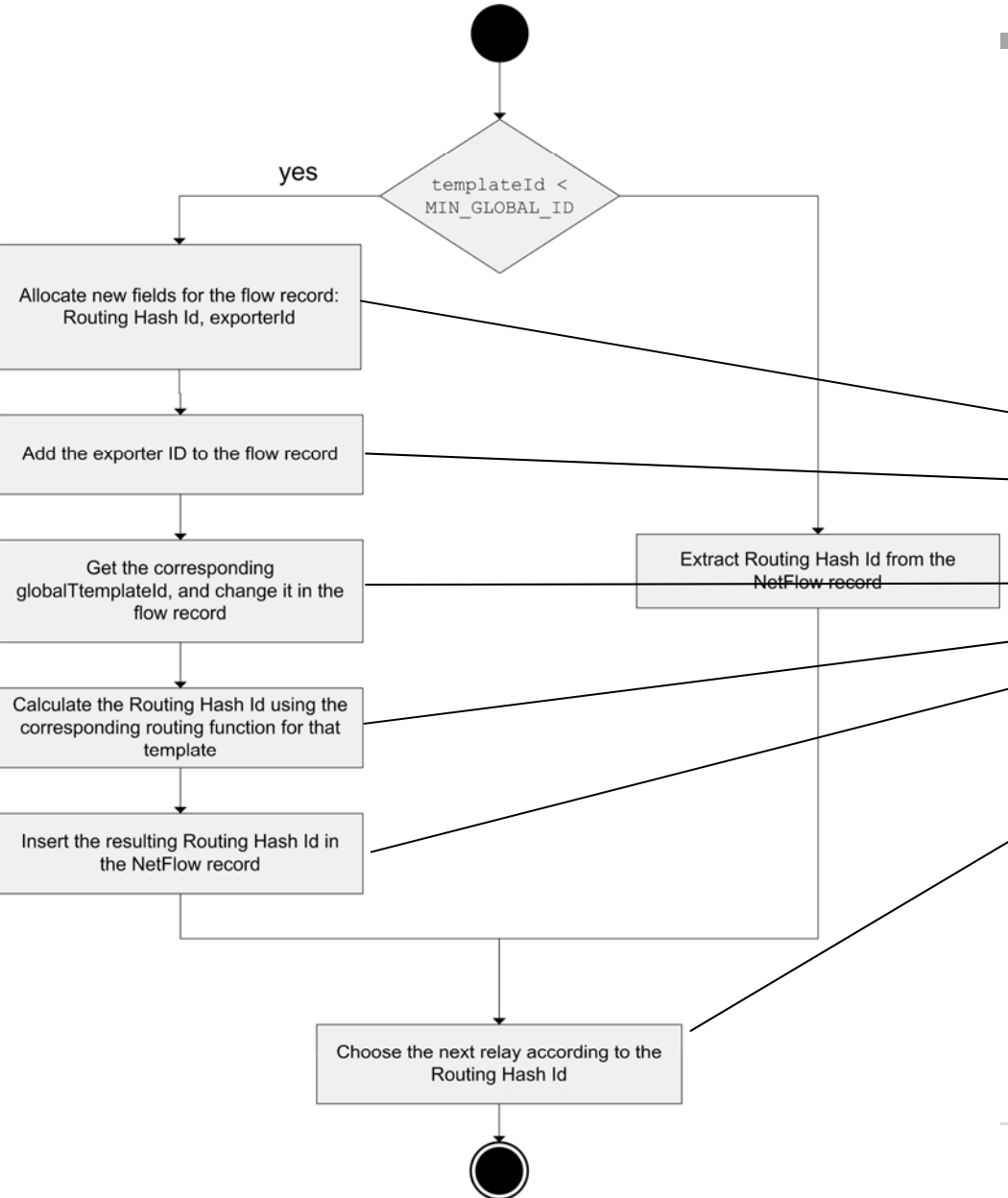
- *These operations are performed by the first SCRIPT node to receive a flow record*

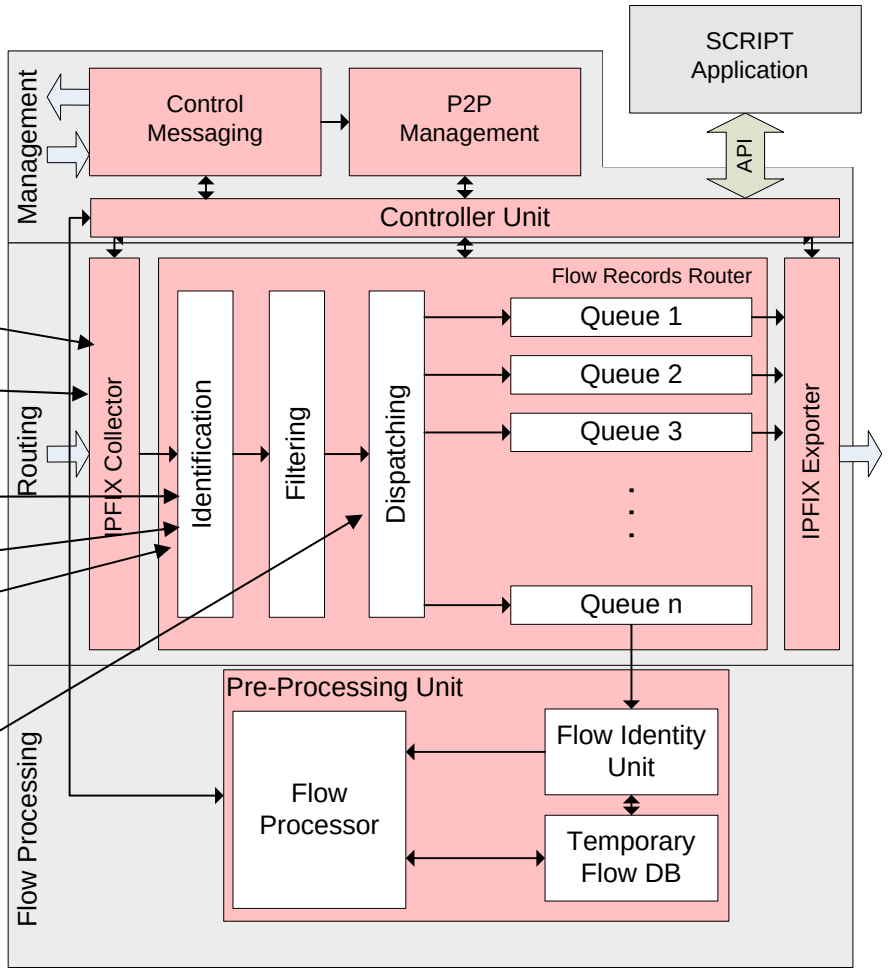
Backward Compatibility

- NetFlow v5 can be also handled by SCRIPT
- A special template exists for the NetFlow v5 format
- At the first node, the v5 records are transformed to IPFIX
- From there they will be treated as IPFIX records
- NetFlow v5 only works with UDP



Flow Record Routing Process





Temporary Flow Handling

- Nodes may be removed
  - Upgrades / Reconfiguration / Power outages, etc.
- Node removal can be permanent / temporary
- Permanent removal:
  - The node is excluded from the P2P overlay
  - Responsibility for its share of flow records is shifted to his neighbor
  - Next node to join the SCRIPT network will receive the identity of the removed node
- Temporary removal
  - Node is kept in the overlay but marked unavailable
  - Flow records are redistributed to other SCRIPT nodes for temporary storage
  - After some time those flow records are re-injected in the SCRIPT network

Implementation and Deployment

- Implemented in C++
- ACE and sctp libraries required to compile and run SCRIPT
- Deployed and tested on UBUNTU / Debian / Fedora Core machines
- Also compiled and deployed on Cisco AXP (Application Extension Platform)
  - 1 GHz Intel Celeron processor, 512 MB RAM, 80GB HDD
  - On 2811 chassis
  - Firmware v1.1.1 (kernel 2.6.14)
  - No support for sctp enabled in the kernel
  - UDP version tested

Distributed SCRIPT Applications in 5 Steps

- Step 1: An application class has to be implemented

  ```
  class MyApp : public LocalProcessor {…}
  ```

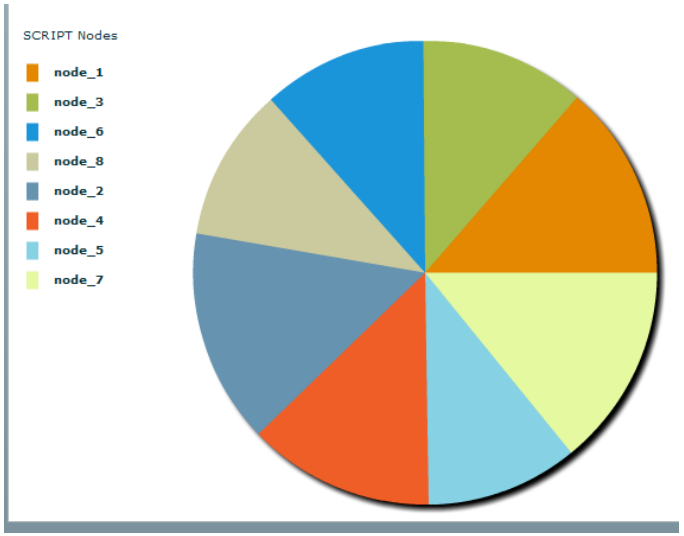
- Step 2: Overwrite method *`_process(sc_flowRecord *fr);`*
  - This method is called every time the SCRIPT node receives a flow record that is needed by the application MyApp
- Step 3 (optional): Overwrite method *`notify(u_char*);`*
  - This method is used to send application-specific messages to the application
  - E.g. queries, configuration parameters
- Step 4: Instantiate a SCRIPT Engine and an application object

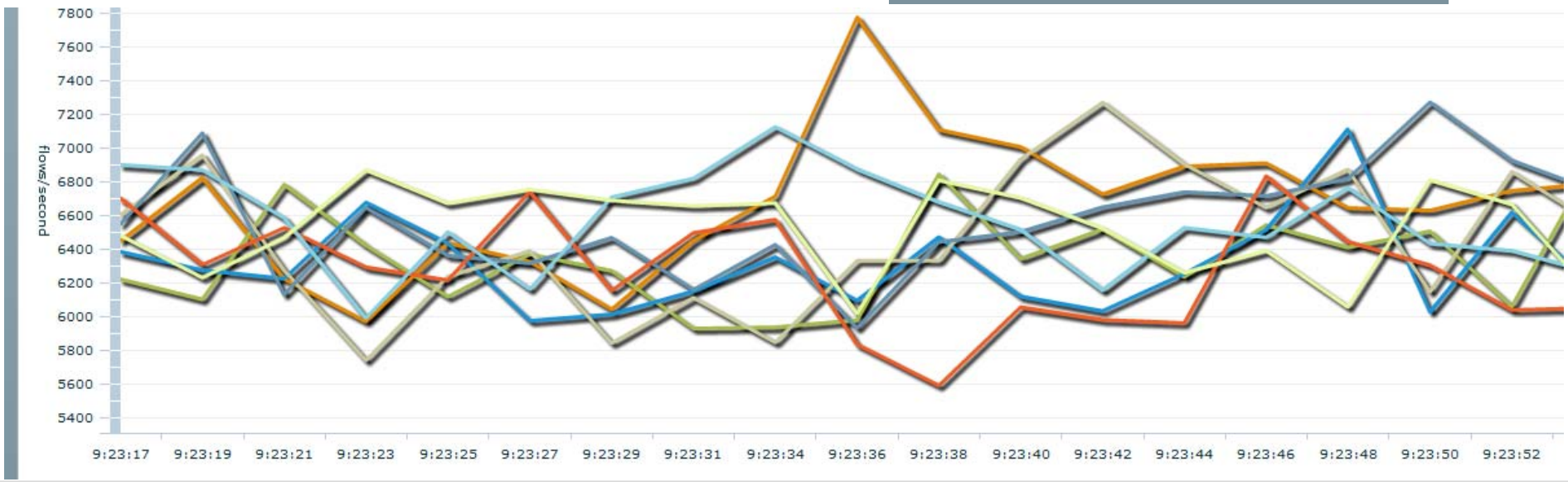
  ```
  ScriptEngine se(localPort, collectorPort, bootstrapIP, bootstrapPort)
  MyApp *app1 = new MyApp(myApplicationId);
  ```

- Step 5: Register the application to SCRIPT Engine
  - Register once for each “interesting” template
  - `se.registerApplication(globalTemplateID1,app1);`
  - `se.registerApplication(globalTemplateID2,app1);`

SCRIPT Monitoring Tool

- Adobe Flex application
- Shows the load on each node
- Shows the distribution of flow records to all the nodes
- Can be easily extended to support other parameters





Summary and Conclusions

- SCRIPT is a platform for scalable IPFIX data distribution
  - Based on a P2P overlay
  - Allows load balancing for processing of flow records
  - Scalable increase of storage space for flow records
  - Faster flow query response for large flow repositories
- Simple API to deploy monitoring applications in a distributed environment
- Extensible due to modular design
  - Example of implemented extensions:
    - Temporary handling of flow records
    - Locality
- Design and implementation independent of the monitoring application
- Deployable without change of exporters
- Can receive IPFIX as well as NetFlow v9/v5 records