ForCES: An implementation

Evangelos Haleplidis (ehalep@ece.upatras.gr) (Speaker)
Odysseas Koufopavlou (odysseas@ece.upatras.gr)
Denazis Spyros (sdena@ece.upatras.gr)
University of Patras
Department of Electrical and Computer Engineer
IETF 70 - Vancouver
December 4, 2007
Presentation Summary

- Driving ForCES
- Protocol/Model Implementation
- Developing ForCES
- Questions
Driving ForCES (1)

- **Flexinet FP6-IST1 507646**
  - Scalable and modular network architecture offering cross-connect control, switching/routing control, and advanced services management/access functions at the network access points.

- **ForCES Implementation consists of:**
  - Heartbeats, Association Messages, Simple Configuration Messages, Simple Query Messages. (code based on protocol draft ver06)
  - LFBs:
    - FEProtocolLFB.
    - FEObjectLFB.
    - Incoming LFB.
    - Outgoing LFB.
    - Classifier LFB.
Driving ForCES (2)

- Phosphorus FP6 034115¹
  - Enhance and demonstrate solutions that facilitate vertical and horizontal communication among applications middleware, existing Network Resource Provisioning Systems, and the proposed Grid-GMPLS Control Plane.
  - UvA² & UoP part: Token Based Switch (TBS) is a low-level system for traffic routing at high speeds based on packet authentication.
    - UvA partners: Mihai Cristea³, Yuri Demchenko⁴.

¹http://www.ist-phosphorus.eu/
²University of Amsterdam
³cristea@science.uva.nl
⁴demch@science.uva.nl
Current protocol implementation consists of:

- 1 CE (ForCEG: ForCES Gateway)
- 1 FE containing:
  - FEP protobuf LFB.
  - FEO object LFB.
  - Rx LFB.
  - Tx LFB.
  - Token Switch LFB.
  - Token Builder LFB.

TML:
- TCP/IP.
Protocol/Model Implementation (2)

- Protocol Parts implemented as yet:
  - Association messages.
  - Heartbeat messages.
  - SET messages.
  - GET messages.
Protocol/Model Implementation (3)

- CE (ForCEG):
  - Basic CE functionality.
  - Incorporates a Web Service for sending commands.
  - Commands are processed in XML and translated in ForCES.
FE:

- Intel IXP enp2611 (w/o security) / 2850 (w security).
  - Mihai Cristea (cristea@science.uva.nl) (TBS development & IXP coding)

Token Based Switch

- Low-level system for traffic routing at high speeds based on packet authentication.

Two major entities:

- Token Builder (TB): Computes a token and insert it into the packet.
- Token Switch (TS):
  - Receives a packet with a token.
  - Computes a local token (as TB)
  - Compares tokens. (if match the packet is authorised)
TBS:
- Mapping of TB and TS varies on the application type.
  - Entrance point of a packet: Only TB.
  - Entrance point to an authorized network: Only TS.
  - Internal change of authorized network (need to prepare a new token): TS & TB.
- One configurable FE.
## Rx Model:

### Data Types:

<table>
<thead>
<tr>
<th>name</th>
<th>synopsis</th>
<th>at:atomic</th>
<th>struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Values that can be applied to incoming and outgoing ports for condition-forwarding.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### LPBCClassRef:

<table>
<thead>
<tr>
<th>name</th>
<th>synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxLFB</td>
<td>The port RxLFB for the FEL-Tvs2.</td>
</tr>
</tbody>
</table>

### Components:

<table>
<thead>
<tr>
<th>name</th>
<th>synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>The condition upon which to go to</td>
</tr>
<tr>
<td>IngressPort</td>
<td>The Ingress Port</td>
</tr>
<tr>
<td>OutgressPort</td>
<td>The Outgress Port</td>
</tr>
</tbody>
</table>

### Data Values:

<table>
<thead>
<tr>
<th>name</th>
<th>synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-only</td>
<td>A value with condition forwarding</td>
</tr>
<tr>
<td>Packet_Count</td>
<td>The number of packets that are coming into the network.</td>
</tr>
</tbody>
</table>
### TB Model:

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>synopsis</th>
<th>struct</th>
<th>array</th>
<th>atomic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tuple</td>
<td>A Tuple from the TVS Ticket</td>
<td>struct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Active Tuple</td>
<td>An Array with the Active Tuples</td>
<td></td>
<td>array</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Ports</td>
<td>Values that can be applied to incoming and outgoing Port for Condition Forwarding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ConditionForwarding</td>
<td>A Condition Forwarding Case. Has incoming and outgoing port</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ActiveTupleCount</td>
<td>An Active Tuple with counters</td>
<td>struct</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### LFClassDef

<table>
<thead>
<tr>
<th></th>
<th>LFClassID</th>
<th>name</th>
<th>synopsis</th>
<th>version</th>
<th>components</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>TBLto</td>
<td>Token Builder TBL</td>
<td>1.0</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>componentID</th>
<th>access</th>
<th>name</th>
<th>synopsis</th>
<th>typeRef</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>read-write</td>
<td>Tuples</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>read-write</td>
<td>Condition_FWD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>read-read</td>
<td>Packet_Count</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## TS Model:

### Data Types:

<table>
<thead>
<tr>
<th>Name</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuple</td>
<td>A Tuple from the TS Ticket</td>
</tr>
<tr>
<td>ActiveTuple</td>
<td>An Array with the Active Tuples and their counters</td>
</tr>
<tr>
<td>ActiveTupleCount</td>
<td>An Active Tuples with its counters</td>
</tr>
<tr>
<td>Ports</td>
<td>Values that can be applied to incoming and outgoing Port for ConditionForwarding</td>
</tr>
<tr>
<td>ConditionForwarding</td>
<td>A Condition Forwarding Case, Has incoming port and outgoing port</td>
</tr>
</tbody>
</table>

### LPBClassDef

<table>
<thead>
<tr>
<th>ComponentID</th>
<th>Component</th>
<th>Access</th>
<th>Name</th>
<th>Synopsis</th>
<th>TypeRef</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>EventID</th>
<th>Event</th>
<th>Name</th>
<th>Synopsis</th>
<th>EventTarget</th>
<th>EventCreated</th>
<th>EventReports</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>EndCountSurpassed</td>
<td>If the End Count is surpassed the UFB will send a message to the CE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Protocol/Model Implementation (8)

- **Tx Model:**

  ```plaintext
  dataTypeDefs
  dataTypeDef
  name Ports
  synopsis Values that can be applied to Incoming and Outgoing Port for Condition Forwarding
  atomic
  baseType u8
  rangeRestriction
  allowedRange
  = max 0
  = min 9

  LFBClassDefs
  LFBClassDef
  LFBClassID 4
  name TxLFB
  synopsis The port Tx LFB for the FE - TVS
  version 1.0
  components
  component (2)
  componentID access name synopsis typedef
  1 1 read-only OutputPort The output port of IXP Ports
  2 2 read-reset Packet_Count The number of packets that are leaving the IXP uint32
  ```
Developing ForCES (1)

- Challenges encountered:
  - Hardware not ForCES compatible.
  - Complex Model Components.
  - Dynamic Protocol Messages.
  - Protocol Interface.
Developing ForCES (2)

- FE
  - **Java (ver6)**
    - **Pro’s:**
      - Easier to code and handle.
      - No use of pointers.
    - **Con’s**
      - Different variable types. (e.g. There are no unsigned types!)
      - No system calls for use with hardware.
  - **C++ Code (ver11)**
    - **Pro’s**
      - System calls.
    - **Con’s**
      - Pointer usage may cause problems.
Developing ForCES (3)

- FE:
  - Each Component has a function that sets/gets/dels actual hardware values.
  - Easy to create LFB Components based on hierarchy.

```
Basic Component
  ├── Byte
  │    └── Byte Array
  ├── Short
  │    └── Short Array
  ├── Integer
  │    └── Integer Array
  └── Double
      └── Double Array
```
LFB Component Hierarchy:

<table>
<thead>
<tr>
<th>Component</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>protected int m_ComponentID</td>
</tr>
<tr>
<td></td>
<td>protected bool m_bR_W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public Component(int CompID, bool RW)</td>
</tr>
<tr>
<td></td>
<td>public int get_ComponentID()</td>
</tr>
<tr>
<td></td>
<td>public char* getValue(char* data)</td>
</tr>
<tr>
<td></td>
<td>public char* setValue(char* data)</td>
</tr>
<tr>
<td></td>
<td>public char* getSingleValue(ILV* lv)</td>
</tr>
<tr>
<td></td>
<td>public char* setSingleValue(ILV* lv)</td>
</tr>
<tr>
<td></td>
<td>public char* delValue(char* data)</td>
</tr>
<tr>
<td></td>
<td>public char* delSingleValue(ILV* lv)</td>
</tr>
<tr>
<td></td>
<td>protected char* getHardwareValue()</td>
</tr>
<tr>
<td></td>
<td>protected bool setHardwareValue()</td>
</tr>
<tr>
<td></td>
<td>protected bool delHardwareValue()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component Byte</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>private char m_cByte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component Byte</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public Component_Bye(int CompID, bool RW, char data)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component_BYTE_ARRAY</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>private HashMap&lt;integer, Component_BYTE&gt; Bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component_BYTE_ARRAY</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public Component_BYTE_ARRAY(int CompID, bool RW)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component_BYTE_ARRAY</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public Component_BYTE_ARRAY(int CompID, bool RW)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component_BYTE_ARRAY</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public Component_BYTE_ARRAY(int CompID, bool RW)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component_BYTE_ARRAY</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public Component_BYTE_ARRAY(int CompID, bool RW)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component_BYTE_ARRAY</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public Component_BYTE_ARRAY(int CompID, bool RW)</td>
</tr>
</tbody>
</table>
**Developing ForCES (5)**

- **Complex LFB’s:**

  ```java
  public class Component {
      protected int m_iComponentID;
      protected bool m_bR_W;
      public Component(int ComplID, bool RW) {
          // Constructor code here
      }
      public int get_iComponentID() {
          // Getter method code here
      }
      public char* getValue(char* data) {
          // Value getter method code here
      }
      public char* setValue(char* data) {
          // Value setter method code here
      }
      public char* getSingleValue(ILV* ilv) {
          // Single value getter method code here
      }
      public char* setSingleValue(ILV* ilv) {
          // Single value setter method code here
      }
      public char* setValue(char* data) {
          // Value getter method code here
      }
      public char* setValue(char* data) {
          // Value setter method code here
      }
      protected bool setHardwareValue() {
          // Hardware value setter method code here
      }
      protected bool delHardwareValue() {
          // Hardware value deleter method code here
      }
      protected char* getHardwareValue() {
          // Hardware value getter method code here
      }
  }
  
  public class Component_Array {
      protected Map<int, Component> Components;
      public Component_Array(int ComplID, bool RW) {
          // Array constructor code here
      }
      public void Add_Component(Component* Comp) {
          // Add component method code here
      }
      public int get_iComponentID() {
          // Array value getter method code here
      }
      public char* getValue(char* data) {
          // Array value getter method code here
      }
      public char* setValue(char* data) {
          // Array value setter method code here
      }
      public char* getSingleValue(ILV* ilv) {
          // Array single value getter method code here
      }
      public char* setSingleValue(ILV* ilv) {
          // Array single value setter method code here
      }
      public char* setValue(char* data) {
          // Array value getter method code here
      }
      public char* setValue(char* data) {
          // Array value setter method code here
      }
      protected bool setHardwareValue() {
          // Array hardware value setter method code here
      }
      protected bool delHardwareValue() {
          // Array hardware value deleter method code here
      }
      protected char* getHardwareValue() {
          // Array hardware value getter method code here
      }
  }
  ```
Developing ForCES (6)

- Protocol Basic Messages

<table>
<thead>
<tr>
<th>Basic_TLV</th>
<th>ILV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attributes</strong></td>
<td><strong>Attributes</strong></td>
</tr>
<tr>
<td>protected short m_shType</td>
<td></td>
</tr>
<tr>
<td>protected short m_shLength</td>
<td></td>
</tr>
<tr>
<td>protected char* m_pData</td>
<td></td>
</tr>
<tr>
<td>protected List&lt;Basic_TLV&gt; m_INextTLV</td>
<td></td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td><strong>Operations</strong></td>
</tr>
<tr>
<td>public Basic_TLV(  )</td>
<td></td>
</tr>
<tr>
<td>public Basic_TLV( char* raw_tlv )</td>
<td></td>
</tr>
<tr>
<td>public Basic_TLV( short Type, short Length, char* Data )</td>
<td></td>
</tr>
<tr>
<td>public short get_shType(  )</td>
<td></td>
</tr>
<tr>
<td>public short get_shLength(  )</td>
<td></td>
</tr>
<tr>
<td>public char* get_pData(  )</td>
<td></td>
</tr>
<tr>
<td>public void addTLV(  )</td>
<td></td>
</tr>
<tr>
<td>public Basic_TLV* get_NextTLV(  )</td>
<td></td>
</tr>
<tr>
<td>public void replace_previousTLV( Basic_TLV* tlv )</td>
<td></td>
</tr>
<tr>
<td>public int Calculate_TLVs_length(  )</td>
<td></td>
</tr>
<tr>
<td>public char* serialize_me(  )</td>
<td></td>
</tr>
<tr>
<td>private int m_iIndex</td>
<td></td>
</tr>
<tr>
<td>private int m_iLength</td>
<td></td>
</tr>
<tr>
<td>private char* m_pData</td>
<td></td>
</tr>
<tr>
<td>public ILV(  )</td>
<td></td>
</tr>
<tr>
<td>public ILV( char* raw_iLV )</td>
<td></td>
</tr>
<tr>
<td>public ILV( int Index, int Length, char* Data )</td>
<td></td>
</tr>
<tr>
<td>public int get_index(  )</td>
<td></td>
</tr>
<tr>
<td>public int get_iLength(  )</td>
<td></td>
</tr>
<tr>
<td>public char* get_pData(  )</td>
<td></td>
</tr>
<tr>
<td>public char* serialize_me(  )</td>
<td></td>
</tr>
</tbody>
</table>
Developing ForCES (7)

- Protocol Message Hierarchy

```
forCES
Protocol Message Hierarchy

Basic_TLV

Attributes
protected short m_shType
protected short m_shLength
protected char* m_pdata
protected List<Basic_TLV> m_iNextTLV

Operations
public Basic_TLV( )
public Basic_TLV( char* raw_tv )
public Basic_TLV( short Type, short Length, char* Data )
public short get_shType( )
public short get_shLength( )
public char* get_pdata( )
public void addTLV( )
public Basic_TLV* get_iNextTLV( )
public void replace_previousTLV(Basic_TLV* tv)
public int Calculate_TLVs_length( )
public char* serialize_me( )

LFBSel_TLV

Attributes
private int m_LFBClass_ID
private int m_LFBInstance_ID

Operations
public LFBSel_TLV( )
public LFBSel_TLV( char* raw_tv )
public LFBSel_TLV( short Type, short Length, int LFBClassID, int LFBInstanceID, char* Data )
public int get_LFBClass_ID( )
public int get_LFBInstance_ID( )

public char* serialize_me( )

PathData_TLV

Attributes
private short m_shFlags
private short m_shIDcount
private int* m_pIds

Operations
public PathData_TLV( )
public PathData_TLV( char* raw_tv )
public PathData_TLV( short Type, short Length, short Flags, short IDcount, int* IDs, char* data ) [PathData_TLV]
public short get_shFlags( )
public short get_shIDcount( )
public int* get_pIds( )

public char* serialize_me( )
```

Developing ForCES (8)

- ForCES is a protocol for configuration of the Forwarding Plane.
- What happens when multiple programs need to configure the same FE? Who controls what?
- Multiple CE controlling an FE or a single manageable CE controlling an FE?
- Need for an Open API between CE and programs for issuing commands.
Developing ForCES (9)

- Translates commands from a Generic Web Service API into ForCES packets.
- Conceal ForCES model & protocol from programs.
- Connections of multiple programs into one Forwarding Element.
- Advertise API.
Developing ForCES (10)

- ForCEG Architecture (not fully implemented)
Questions (Ours)

- How will ForCES be used by higher layer applications & to alleviate network services?
- What is the relationship between ForCES and Netconf? Similarities / Differences?
Questions (Yours)

- Any comments are welcome
Thank you!