DCEP-Sim: An Open Simulation Framework for Distributed CEP
Introduction for Users and Prospective Developers

Fabrice Starks

Stein Kristiansen, Thomas Plagemann
Introduction and Motivation

• Data streams and information flow processing
  – Financial tickers
  – Traffic management
  – Internet of Things
  – eHealth

• Real-time processing:
  – Data Stream Management Systems
  – Complex Event Processing
Distributed CEP

• CEP instances communicate via a network
  – End to end delay
  – Error rate
  – Available bandwidth

• How deterministic are the network properties
  – Guaranteed QoS vs. best effort
  – Private vs. public networks
  – Static vs. mobile networks

[Source: https://www.pcsteps.com/10751-mobile-internet-e-3g-h-plus-4g-mobile-network/]
Distributed CEP - Challenges

• Test and evaluate
• Real world vs. emulation vs. simulation
• What are realistic, representative network properties?

Effort to do proper testing and evaluation

The more dynamics, the harder
Some insights from a recent survey


- 13 publications on mobile Distributed CEP with 19 evaluation reports
  - 2 based on mathematical modeling
  - 3 based on PlanetLab experiments
  - 3 based on emulation
  - 11 based on simulation
    - 7 based on simulators created for the specific experiments
    - 4 based on popular network simulators (J-Sim, OMNet and PeerSim)

- The missing consensus on evaluation approaches motivated our development of DCEP-Sim (presented at DEBS 2017)
Aim of this tutorial

• For us:
  – Start an open source project with DCEP-Sim

• For you (assuming 3 types of attendees):
  – Explain what you could do with DCEP-Sim in your work
  – How to get started with DCEP-Sim
  – How to use DCEP-Sim in your research and contribute to the code base
Disclaimer

• DCEP-Sim is
  – not a commercial product,
  – but an outcome of the PhD thesis from Fabrice Starks
  – and is now open to contributions from the community

• DCEP-Sim inherits strength and weaknesses of ns-3
  – many high quality network models
  – high flexibility
  – powerful tracing and data collection
  – efficient
  – software execution time is not considered
Outline

• Introduction and motivation
  • Concepts and architecture of the distributed CEP engine in DCEP-Sim
    – Requirements
    – Design principles
    – CEP engine
    – Placement
    – Overall architecture
  • Introduction to the network simulator ns-3
    – Principles of discrete event simulation
    – ns-3 Overview
    – Key ns-3 modeling and simulation concepts
    – Fundamental ns-3 models
    – ns-3 simulation via example
Outline (cont.)

• DCEP-Sim use and extensions
  – Overview code structure
  – How do I run DCEP-Sim & how works a «script»
  – Changing the workload
  – How are placement policies implemented -> adding new placement
  – How are operators implemented -> adding new operators

• Conclusions

• Hands-on if you want to install ns-3 and run DCEP-Sim on you Linux laptop
Placement the Main Challenge of Distributed CEP

Query: \((A \lor B) \land C\)
Placement the Main Challenge of Distributed CEP

Where to place the operators?

Network link properties & overlay link properties:
  Latency, available bandwidth, loss

Traffic properties:
  High event rate vs. low event rate from sources
  Selectivity of operators

Other concerns: node resources, constraints, security
Placement the Main Challenge of Distributed CEP

What do you do if you have some cool new ideas for placement?

Model, design, implement

Test & implement – but how? → DCEPSim
DCEP-Sim Goals

- Tool for experimentation with Distributed CEP solutions
- Realistic models of various network types and conditions
- Ability to create arbitrary traffic patterns
- Support CEP query and query processing concepts
  - Operators, windows, selection policy, consumption policy
  - without the need to implement a »full CQL»
- Extensibility and flexibility
- Easy to use
Major Design Decisions

• Use the well established network simulator ns-3
  – Benefit from many years effort
  – Many existing models for link, network, transport level protocols,
    ++
  – High degree of realism
  – Tools for debugging, tracing, data collection, ++

• Simulation instead of emulation
  – Scalability
Engineering Principles

• Separation of concerns

• Separation of mechanisms and policies
Design & Implementation Approach

• Start:

• Apply the engineering principles to develop the architecture

• Components & sub-components are good candidates to be implemented as objects

• Leverage the ns-3 features for the implementation of an extensible and flexible solution
Functional Architecture of an IFP System

[Image of a diagram showing the architecture with components like Receiver, Decider, Knowledge Base, Producer, Forwarder, Clock, History, Seq, and Rules]

[Cugola et al. 2012]
DCEP-Sim Components

- CEP Engine
- Source
- Sink
- Placement
- Dispatcher
- Communication
CEP-Engine

Detector → Producer → Forwarder

CEP Engine

Placement

Source

Dispatcher

Sink

Communication
DCEP-Sim
Components

CEP Engine

Placement

Source

Sink

Dispatcher

Communication
Forwarder vs. Communication

We do not want to change the CEP engine to use different protocols, etc.!
Forwarder & Placement

Detector → Producer → Forwarder

Forwarder passes events to Placement

Placement uses Event Routing Table to determine destination of event

Event Routing Table

CEP Engine

Source → Sink

Placement

Dispatcher

Communication
Operator in Detector
Placement

• Assign operators to event brokers
  – Initial
  – Adaptation
  – Challenging optimization problems
    • Network utilization
    • Energy consumption
    • Event delivery latency
    • (security) constraints
• Result of placement: Operator overlay resp. operator tree
• Further tasks: event routing & forwarding
Example: Centralized Placement

- Sink node knows network topology
- Could calculate optimal placement for \((A \lor B) \land C\)
- Sends the operators to the selected brokers
- Sends routing information to all overlay nodes
Example: Centralized Placement as it is in the Code

• Places the entire query on one node
• Sends the operators to the selected broker
• Sends routing information to all overlay nodes
Example: Distributed Placement

- Sink (CCC) forwards operator graph on the shortest path towards sources
- On each following node:
  - can all sources reached through a single link?
    - Yes: forward entire (sub-)graph
    - No: split operator graph, place operator locally forward sub-graphs, update event routing table

Starks, F., Plagemann, T.: *Operator placement for efficient distributed complex event processing in MANETs*, WiMOB 2015
Placement

Event routing table

Placement Policy

Placement Mechanism
Source and Sink

Source

Produce atomic events

Sink

Pose a query

Receive event stream
Communication

• Responsible for transport of messages
  – Placement messages
    • Forwarding of (parts of) operator graph
    • Coordination of placement adaptation
  – Event notifications
• Current implementation uses UDP
Dispatcher

- Facade component
- Dispatches
Component Interactions
Centralized Placement
Component Interactions
Distributed Placement
Component Interactions
Event Processing
### Scalability: number of brokers

<table>
<thead>
<tr>
<th>Number of brokers</th>
<th>Brokers</th>
<th>Events</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>500</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The graph shows the relationship between the number of nodes and the time in milliseconds for different scenarios.
Scalability: number of events

<table>
<thead>
<tr>
<th>Number of events</th>
<th>brokers</th>
<th>operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
End of Part 1

• Very short motivation for distributed CEP
• Design approach
• Components and their responsibility
• Component interaction

• Components correspond to objects in the code (part 3)
• To understand the implementation it is very important to understand ns-3 (next part)
Outline (cont.)

- DCEP-Sim use and extensions
  - Overview code structure
  - How do I run DCEP-Sim & how works a «script»
  - Changing the workload
  - How are placement policies implemented -> adding new placement
  - How are operators implemented -> adding new operators

- Conclusions

- Hands-on if you want to install ns-3 and run DCEP-Sim on you Linux laptop
DCEP-Sim on github

- [https://github.com/fabricesb/DCEP-Sim](https://github.com/fabricesb/DCEP-Sim)
- GNU GPLv2 license (to be in line with ns-3)
<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>made some clean up and added one additional simulation program</td>
<td>3 days ago</td>
</tr>
</tbody>
</table>

DCEP-Sim code
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>antenna</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>aodv</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>applications</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>bridge</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>brite</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>buildings</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>click</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>config-store</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>core</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>csma-layout</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>csma</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>dcep</td>
<td>made some clean up and added one additional simulation program</td>
<td>3 days ago</td>
</tr>
<tr>
<td>dsdv</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>dsr</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>energy</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>fd-net-device</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>flow-monitor</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>internet-cast</td>
<td></td>
<td>26 days ago</td>
</tr>
</tbody>
</table>

**DCEP-Sim code**

<table>
<thead>
<tr>
<th>Branch: master</th>
<th>DCEP-Sim / src /</th>
</tr>
</thead>
<tbody>
<tr>
<td>fabricesb</td>
<td>made some clean up and added one additional simulation program</td>
</tr>
</tbody>
</table>
Here you find the example script we will walk through named decep-example.cc

Here you find all models, i.e., the core of DCEP-Sim
Components, objects, and aggregation

DCEP application
- Communication
- Placement
- Data source
- Sink
- CEP engine

Inside dcep.cc

Communication.cc

Placement.cc

dcep.cc

cep-engine.cc
Components, objects, and aggregation (cont.)

All in cep-engine.cc

```cpp
CEP::CEPEngine()
{
    Ptr<Forwarder> forwarder = CreateObject<Forwarder>();
    Ptr<Detector> detector = CreateObject<Detector>();
    Ptr<Producer> producer = CreateObject<Producer>();
    AggregateObject(forwarder);
    AggregateObject(detector);
    AggregateObject(producer);
}
```
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>cep-engine.cc</td>
<td>made some clean up and added one additional simulation program</td>
<td>3 days ago</td>
</tr>
<tr>
<td>cep-engine.h</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>common.h</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>communication.cc</td>
<td>made some clean up and added one additional simulation program</td>
<td>3 days ago</td>
</tr>
<tr>
<td>communication.h</td>
<td>made some clean up and added one additional simulation program</td>
<td>3 days ago</td>
</tr>
<tr>
<td>dcep-header.cc</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>dcep-header.h</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>dcep-state.cc</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>dcep-state.h</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>dcep.cc</td>
<td>made some clean up and added one additional simulation program</td>
<td>3 days ago</td>
</tr>
<tr>
<td>dcep.h</td>
<td>made some clean up and added one additional simulation program</td>
<td>3 days ago</td>
</tr>
<tr>
<td>message-types.h</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>placement.cc</td>
<td>made some clean up and added one additional simulation program</td>
<td>3 days ago</td>
</tr>
<tr>
<td>placement.h</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>resource-manager.cc</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>resource-manager.h</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>seq-ts-header.cc</td>
<td></td>
<td>26 days ago</td>
</tr>
<tr>
<td>seq-ts-header.h</td>
<td></td>
<td>26 days ago</td>
</tr>
</tbody>
</table>
Right now it contains one helper to set up dcep-app
dcep-app-helper.cc
Typical elements of a script

- Configuration of logging level
- Create and install network topology
- Setting up network, and transport layer
- Setting up the Distributed CEP overlay
- Running the simulation
Create and install network topology

Setting up network, and transport layer

Setting up the Distributed CEP overlay

Running the simulation

LogComponentEnable ("Placement", LOG_LEVEL_INFO);
LogComponentEnable ("Dcep", LOG_LEVEL_INFO);
LogComponentEnable ("Detector", LOG_LEVEL_INFO);
LogComponentEnable ("Communication", LOG_LEVEL_INFO);
uint32_t numNodes = gridWidth*gridWidth;
NodeContainer n;
n.Create (numNodes);

NetDeviceContainer devices =
    SetupWirelessNetwork(n);
MobilityHelper mobility;

mobility.SetPositionAllocator
    ("ns3::GridPositionAllocator", "MinX",
     DoubleValue (0.0), "MinY", DoubleValue (0.0),
     "DeltaX", DoubleValue (distance), "DeltaY",
     DoubleValue (distance), "GridWidth",
     UintegerValue (gridWidth), "LayoutType",
     StringValue ("RowFirst"));

mobility.SetMobilityModel
    ("ns3::ConstantPositionMobilityModel");
mobility.Install (n);
Typical elements of a script

1. Configuration of logging level
2. Create and install network topology
3. Setting up network, and transport layer
4. Setting up the Distributed CEP overlay
5. Running the simulation

```c
OlsrHelper olsr;
InternetStackHelper internet;
internet.SetRoutingHelper (olsr);
internet.Install (n);
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer iface =
  ipv4.Assign (devices);
```
sinkAddress = Address(iface.GetAddress (0));
DcepAppHelper dcepApphelper;
ApplicationContainer dcepApps =
    dcepApphelper.Install (n);
uint32_t eventCode = 1;
for(uint32_t i = 0; i <= numNodes; i++) {
    dcepApps.Get(i)->setAttribute
        ("SinkAddress", AddressValue(sinkAddress));
    dcepApps.Get(i)->setAttribute("placement
        policy", StringValue(placementPolicy));
    if(i == 0) { /* sink node*/
        dcepApps.Get(i)->setAttribute
            ("IsSink", BooleanValue(true));
    } else if ((i == (numNodes-1)) || (i == (numNodes-2))){
        dcepApps.Get(i)->setAttribute("IsGenerator",
            BooleanValue(true));
        dcepApps.Get(i)->setAttribute("event code",
            UintegerValue (eventCode++));
        dcepApps.Get(i)->setAttribute("number of
            events", UintegerValue (numberOfEvents));
    }
Typical elements of a script

1. Configuration of logging level
2. Create and install network topology
3. Setting up network, and transport layer
4. Setting up the Distributed CEP overlay
5. Running the simulation

Code examples:
- `dcepApps.Start (Seconds (1.0));`
- `dcepApps.Stop (Seconds (30.0));`
- `Simulator::Stop(Seconds(35.0));`
- `Simulator::Run ();`
- `Simulator::Destroy ();`
Change the workload

- Current event sources produce uniform traffic
- Configure Distributed CEP instances as data sources in the script, e.g.,

```java
    dcepApps.Get(0)->SetAttribute("IsGenerator", BooleanValue(true));
    dcepApps.Get(0)->SetAttribute("event code",UintegerValue (eventCode++));
```

- Set number of events in the script

```java
    dcepApps.Get(i)->SetAttribute("number of events",
                               UintegerValue (numberOfEvents));
```
Change the workload (cont.)

- Currently, the event rate is set in the `DataSource::GenerateAtomicEvents()` implementation in `dcep.cc`

```cpp
if(counter < numEvents)
{
    Simulator::Schedule (MilliSeconds (100),
                        &DataSource::GenerateAtomicEvents, this);
}
```

Good example of scheduling discrete ns-3 events....

..... to generate at a fixed rate atomic events!
Change workload (cont.)

• For more complex event patterns extend the data source model or create a new data source model
  – Get inspired by ns-3 traffic models
    • Statistical distributions
    • Trace files
    • ........
  – Extend/modify the function `GenerateCEPEEvents()` which can be found in the file `dcep.cc`
Creating Your Own Placement Policy

• Main responsibilities of placement:
  1. Operator assignment
  2. Event routing and forwarding
• Approach:
  – High + low level views
  – Creating a new placement policy
  – Example: centralized placement
Placement Assignment: High-Level Overview

Operator graph

Sink

∧ C
∨ B
∧ A

Locally placed operators
CEP Engine

Remotely placed operators
Communication

Placement Policy

Placement Mechanism

Event routing table

- Next hop
- Output destination

Operator flow

Control flow
Placement Assignment: High-Level Overview

Operator flow

Control flow

Remotely placed operators

Network

Communication
Placement Assignment: High-Level Overview

- Operator graph
- Sink
- Placement Policy
- Placement Mechanism
- Locally placed operators
  - CEP Engine
- Remotely placed operators
  - Communication

Main responsibility of Placement Policy!
The Event Routing Table (ERT)

• Accessed via interface called DcepState
• Important fields in entries:
  – Destination of event (output destination)
  – Destination of the query (next hop)
  – Data sources

• Additional fields mostly for adaptation and monitoring
  – Operator state (active or not)
  – Freeze acknowledgement counter
  – Freeze queue
  – Monitoring
  – Current processor
Data object flow

Schedule execution

Function call

Sink

q = query
o = output address
n = next hop address
e = event

Dcep

Placement

Placement Policy

Placement Mechanism

XPlacementPolicy

CEP Engine

Communication

Event routing table

StartApplication()

Sink

q = query
o = output address
n = next hop address
e = event

Dcep

Placement

DcepState

Communication

DcepEngine
StartApplication() 
BuildAndSendQuery() 
DispatchQuery(q) 
RecvQuery(q) 
DoPlacement() 

q = query
o = output address
n = next hop address
e = event

Data object flow
Schedule execution
Function call
Sink

Placement Policy

Placement Mechanism

CEP Engine

Communication

Data object flow

Schedule execution

Function call

q = query
o = output address
n = next hop address
e = event

StartApplication

BuildAndSendQuery

DispatchQuery

RecvQuery

DoPlacement

Repeated for every operator q

CreateEventRoutingTableEntry

SetOutDest

SetNextHop

ForwardQuery

ERT

StartApplication

BuildAndSendQuery

DispatchQuery

RecvQuery

DoPlacement

ERT

CreateEventRoutingTableEntry

SetOutDest

SetNextHop

ForwardQuery
Sink

Dcep

Placement

Placement Policy

Placement Mechanism

DcepState

XPlacementPolicy

CEP Engine

Communication

Event routing table

StartApplication()!

BuildAndSendQuery()!

DispatchQuery(q)

RecvQuery(q)

DoPlacement()

ActivateDataSource(q)

RecvQuery(q)

SendPacket(q)

ScheduleSend(q)

Action chosen based on decision made by placement policy, which is obtained from the ERT

Repeat for every operator q

Data object flow

Schedule execution

Function call

q = query
o = output address
n = next hop address
e = event
Sink

Dcep

Placement

DcepState

XPlacementPolicy

Communication

Placement Node X

Placement Policy

Placement Mechanism

Placement Node Z

Placement Policy

Placement Mechanism

Network

Data object flow

Schedule execution

Function call

Data object flow

Schedule execution

Function call

q = query
o = output address
n = next hop address
e = event

qe = query
of = output address
ne = next hop address
e = event

Sink

Dcep

Placement

DcepState

XPlacementPolicy

Communication

Placement Node X

Placement Policy

Placement Mechanism

Placement Node Z

Placement Policy

Placement Mechanism

Network

HandleRead(socket)

RecvRemoveMsg(buffer...)

RecvQuery(q)

DoPlacement()

CreateEventRoutingTableEntry(q)

SetOutDest(q,o)

ForwardQuery(q)

ActivateDataSource(q)

RecvQuery(q)

SendPacket(q)

ScheduleSend(q)

if local

if remote

if atomic and local

SendPacket(q)

if remote

ScheduleSend(q)
Event Routing and Forwarding

Data object flow:
- q = query
- o = output address
- n = next hop address
- e = event

Event Routing and Forwarding

Sink
HandleRead(socket)

Dcep
RecvRemoteMsg(buffer, ...)

Placement
RecEvent(e)

DcepState

XPlacementPolicy

Communication

DcepEngine

Schedule execution
Function call

HandleRead(socket)

Sink

Dcep

Placement

DcepState

XPlacementPolicy

Communication

DcepEngine

Schedule execution
Function call
Event Routing and Forwarding

```
if event
SendCepEvent(e)

if not final
ForwardProducedEvent(e)
```

```
Event Routing and Forwarding
```

```
Sink
Dcep
Placement
DcepState
XPlacementPolicy
Communication
```

```
Data object flow
q = query
o = output address
n = next hop address
e = event
```

```
HandleRead(socket)
RecvRemoteMsg(buffer)
RecvCepEvent(e)
ProcessCepEvent(e)
SendFinalEventToSink(e)
```

```
Detector
Producer
CepOperator
Forwarder
```

```
Function call
Schedule execution
Schedule
```
Event Routing and Forwarding

1. HandleRead(socket)
2. DcepEngine
   - RecvRemoteMsg(buffer, ...)
   - RecvCepEvent(e)
     - If event
       - SendFinalEventToSink(e)
         - If final
           - ReceiveFinalEvent(e)
         - Otherwise
           - SendFinalEventToSink(e)
             - If final
               - ReceiveFinalEvent(e)
             - Otherwise
               - SendPacket(e, o)
                 - If remote
                   - ScheduleSend(e, o)

Data object flow:
- q = query
- o = output address
- n = next hop address
- e = event

Schedule execution:
- Function call

Communication:
- CepOperator
- Detector
- Producer
- Forwarder

Object flow:
- q = query
- o = output address
- n = next hop address
- e = event
Adding a New Placement Policy

• Create a sub-class of PlacementPolicy
• Must be defined:
  – Ns-3-specific functions, attributes and trace-sources:
    • Mandatory: GetTypeID() ->
  – configure()
    • Initialisation
  – DoPlacement()
    • Mandatory
    • Manipulate ERT via aggregated DcepState-object ->
  • Call placement mechanism once per operator ->

placement.cc
TypeId CentralizedPlacementPolicy::GetTypeId(void) {
  static Typid tid = Typid("ns3::CentralizedPlacementPolicy")
    .SetParent<PlacementPolicy>()
    .AddConstructor<CentralizedPlacementPolicy> ();
  return tid;
}

dcep-state.h
class DcepState : public Object {
  ...
  void SetNextHop (std::string eventType, Ipv4Address adr);
  void SetOutDest (std::string eventType, Ipv4Address adr);
}

placement.h
class Placement : public Object {
  ...
  void ForwardQuery(std::string eType);
Example Placement Policy: Centralized Placement

Query: $A \lor B$

Operator graph
Example Placement Policy: Centralized Placement

Query: A ∨ B

Operator graph

Sink

Atomic events from A

Atomic events from B

Sources

Final, composite events from v
`void Sink::BuildAndSendQuery() {
    Ptr<Dcep> dcep = GetObject<Dcep>();

    Ptr<Query> q1 = CreateObject<Query>();
    q1->actionType = NOTIFICATION;
    q1->id = query_counter++;
    q1->isFinal = false;
    q1->isAtomic = true;
    q1->eventType = "A";
    q1->output_dest = Ipv4Address::GetAny();
    q1->inevent1 = "A";
    q1->inevent2 = "";
    q1->op = "true";
    q1->assigned = false;
    q1->currentHost.Set("0.0.0.0");
    q1->parent_output = "AorB";
    NS_LOG_INFO ("Setup query " << q1->eventType);
    dcep->DispatchQuery(q1);

    ...

    Ptr<Query> q2 = CreateObject<Query>();
    q2->eventType = "B";
    q2->inevent1 = "B";
    dcep->DispatchQuery(q2);

    ...

    Ptr<Query> q3 = CreateObject<Query>();
    q3->isFinal = true;
    q3->isAtomic = false;
    q3->eventType = "AorB";
    q3->inevent1 = "A";
    q3->inevent2 = "B";
    q3->op = "or";
    NS_LOG_INFO ("Setup query " << q3->eventType);
    dcep->DispatchQuery(q3);
}
void CentralizedPlacementPolicy::configure() {
}

void CentralizedPlacementPolicy::DoPlacement() {
    NS_LOG_INFO("Doing centralized placement");
    Ptr<Placement> p = GetObject<Placement>();
    std::vector<Ptr<Query>>::iterator it;
    std::vector<Ptr<Query>> qs = p->q_queue;
    for (it = qs.begin(); it != qs.end(); ++it) {
        Ptr<Query> q = *it;
        if (!PlaceQuery(q)) {
            Simulator::Schedule(Seconds(3.0), &CentralizedPlacementPolicy::DoPlacement, this);
        } else {
            p->RemoveQuery(q);
        }
    }
}
bool CentralizedPlacementPolicy::PlaceQuery(Ptr<Query> q) {
    Ptr<Placement> p = GetObject<Placement>();
    Ptr<DcepState> dstate = GetObject<DcepState>();
    dstate->CreateEventRoutingTableEntry(q);
    Ptr<Communication> cm = GetObject<Communication>();
    bool placed = false;
    if (!q->isAtomic) {
        dstate->SetNextHop(q->eventType, cm->GetLocalAddress());
        placed = true;
    } else if (q->isAtomic) {
        if (q->eventType == "A") {
            dstate->SetNextHop(q->eventType, Ipv4Address("10.0.0.2"));
            placed = true;
        } else if (q->eventType == "B") {
            dstate->SetNextHop(q->eventType, Ipv4Address("10.0.0.3"));
            placed = true;
        }
    }
    if (placed) {
        NS_LOG_INFO("QUERY PLACED");
        newLocalPlacement(q->eventType);
        if (dstate->GetNextHop(q->eventType).IsEqual(cm->GetLocalAddress())) {
            NS_LOG_INFO("QUERY PLACED ON LOCAL NODE");
            if (!q->isAtomic)
                dstate->SetOutDest(q->eventType, cm->GetLocalAddress());
            else
                dstate->SetOutDest(q->eventType, cm->GetSinkAddress());
        }
    }
    p->ForwardQuery(q->eventType);
    return placed;
}
Add new operators

- Operator implementation based on
- CEP engine wrappers class -> detector class

As mentioned earlier:

the focus for DCEPSim until now was placement → simple event model and few operators implemented
Query vs. Operator

- Query used for placement
- Operator used for event processing

Info managed during placement

The query
- Values to be matched
- Operator

These values are copied into an AndOperator instance
Conceptual structure of operator
Operator class in `cep-engine.cc`

```
CepOperator::

Configure()
Evaluate()
ExpectingEvent()

AndOperator::Configure()
AndOperator::Evaluate()
AndOperator::ExpectingEvent()
```
Event class in `cep-engine.h`

```cpp
class Event : public Object{
public:
    static TypeId GetTypeId (void);
    Event(Ptr<Event>);
    Event();
    void operator=(Ptr<Event>);
    SerializedEvent* serialize();
    void deserialize(uint8_t*, uint32_t);
    uint32_t getSize();
    void CopyEvent (Ptr<Event> e);

    std::string type; //the type of the event
    uint64_t m_seq;
    uint64_t delay;
    uint32_t event_class;
    int32_t hopsCount;
    int32_t prevHopsCount;
};
```
Copy info from query object during placement

Create a buffer manager for the operator

Set consumption and selection policies
Make sure both buffers are not empty

Check which buffer the event belongs to

Find the event from the other buffer which matches the sequence number of the current event
bool AndOperator::ExpectingEvent(std::string eType)
{
    if((event1 == eType) || (event2 == eType))
        return true;
    else
        return false;
}
Conclusions

• DCEPSim is
  – a tool for our research in operator placement for mobile distributed CEP
  – not perfect
  – but «easily» extensible (especially if one gets acquainted with ns-3)

• In case you have any questions/ideas/comments
  – Talk with us here @ DEBS 2018
  – Email us: fabricceb@ifi.uio.no, steikr@ifi.uio.no, plageman@ifi.uio.no