PLAN: A Programming Language for Active Networks

Hicks, Kakkar, Moore, Gunter, Nettles

Anna Philippou

April 1998, University of Pennsylvania

Synopsis

- Motivations and requirements
- The language
- The network
- Implementation
- Summary and conclusions

Motivation

- Introduce programmability into the network

Approach

Packet Level

- programs within packets
- special-purpose language (PLAN)
- network configuration, resource discovery, no need for authentication

Service Level

- programmable network routers
- node-resident services written in a general-purpose language
- authentication may be required for certain services

PLAN basics

- A language for programs carried in packets of a programmable network
- PLAN programs replace the headers of packets in traditional networks
- Ability to call routines written in other languages
- Availability guarantees

Requirements

- Flexibility
- Safety
- Security
- Performance
- Usability

The Language

Functional Language in the style of ML

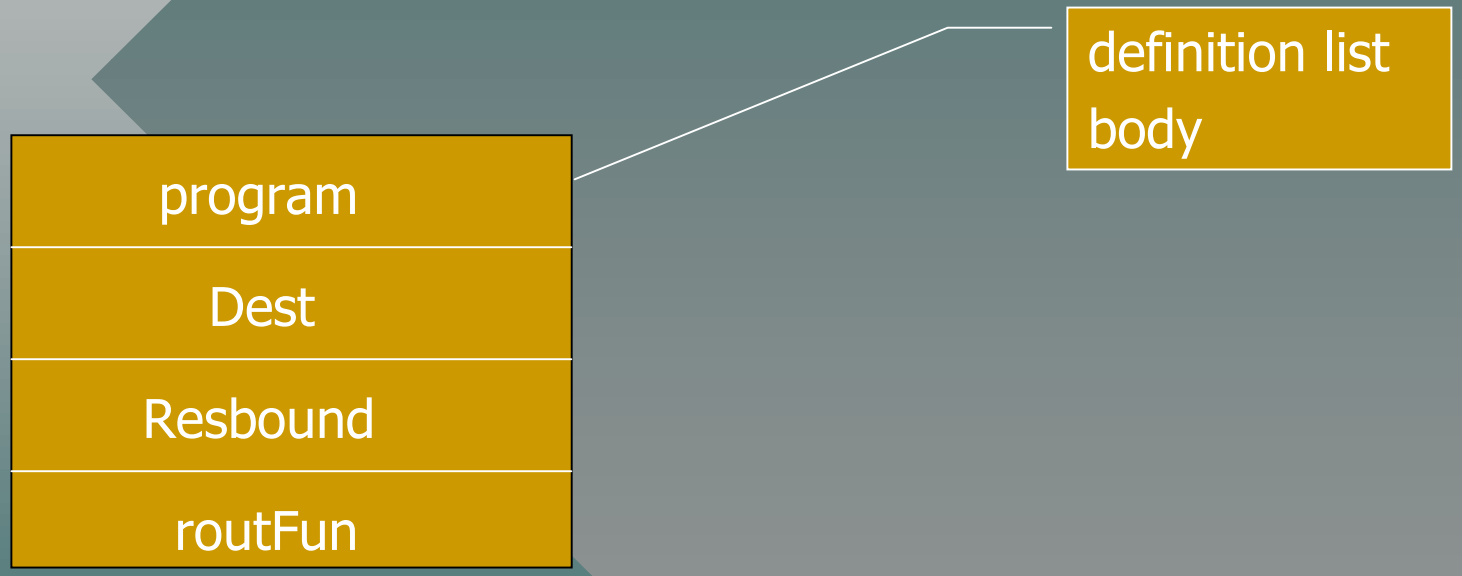
- strictly functional
- strongly typed and statically typeable
- lexically scoped

The Language

However, unlike ML,

- no recursion and unbounded iteration
- no high-order functions
- no pattern matching
- monomorphic
- no mutable state

The PLAN packet



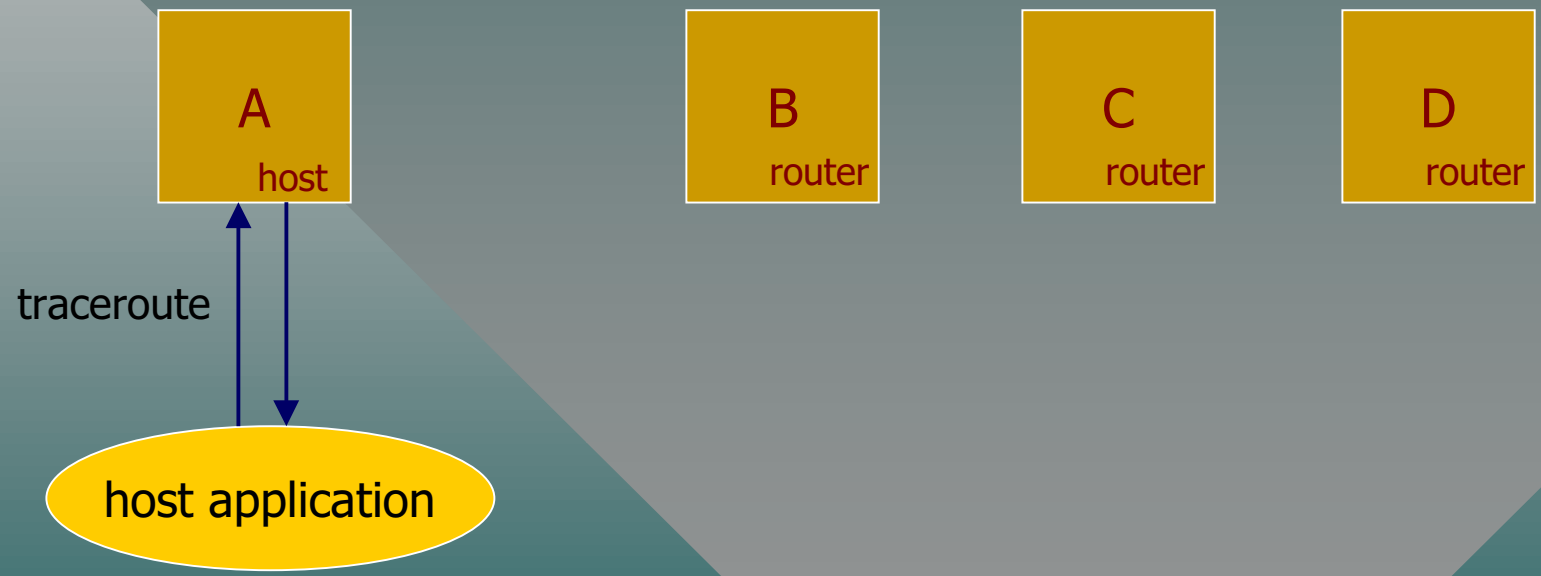
Remote Execution

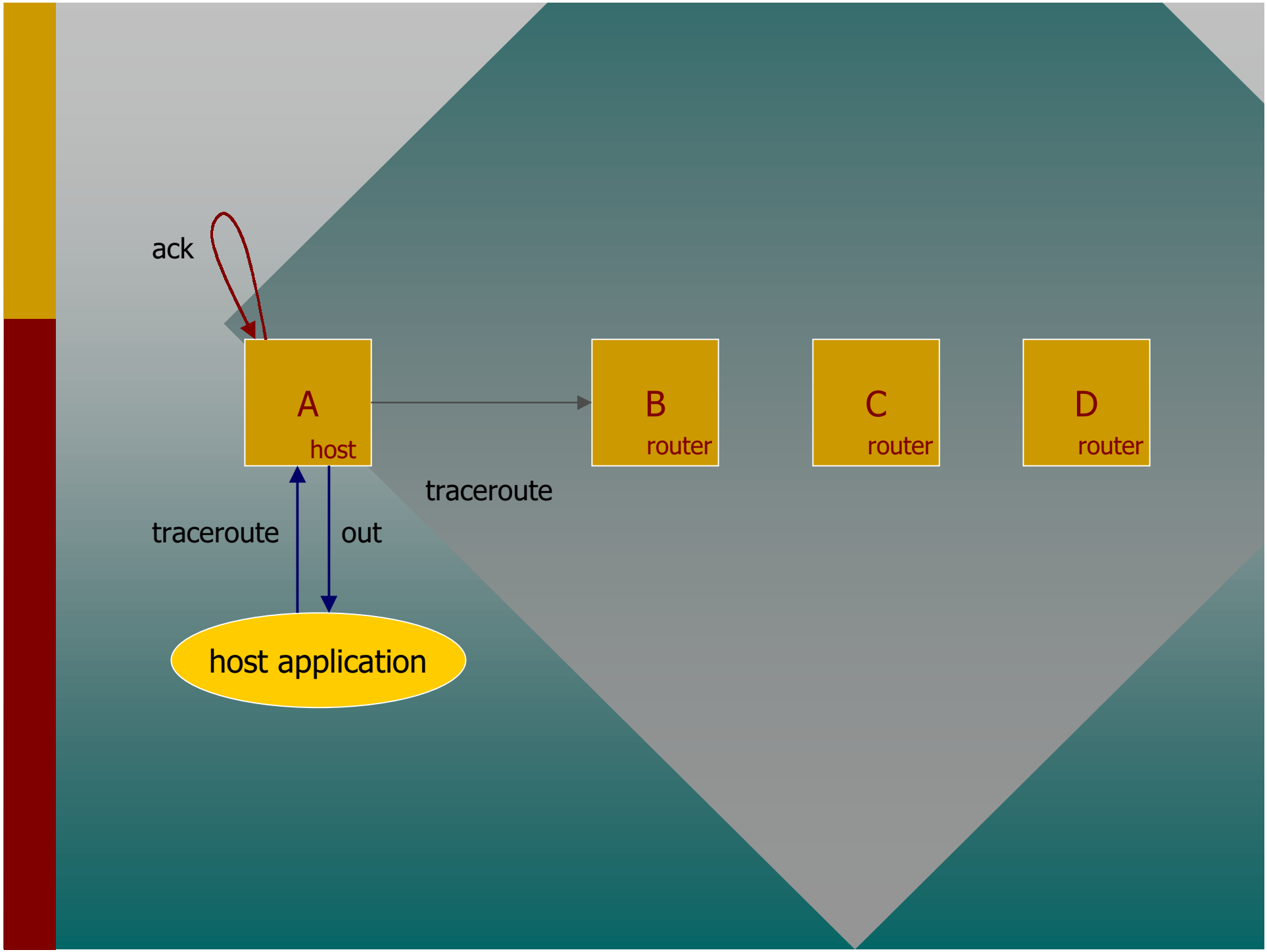
- New packets are created via the PLAN-primitives:
  - OnRemote(prog, dest, ResB, RoutFun)
  - OnNeighbor(prog, dest, ResB, ())
- On creation a packet is sent to its destination to be evaluated.
- Each hop decreases the Resource bound of a packet by one.
- These primitives provide data-transport.

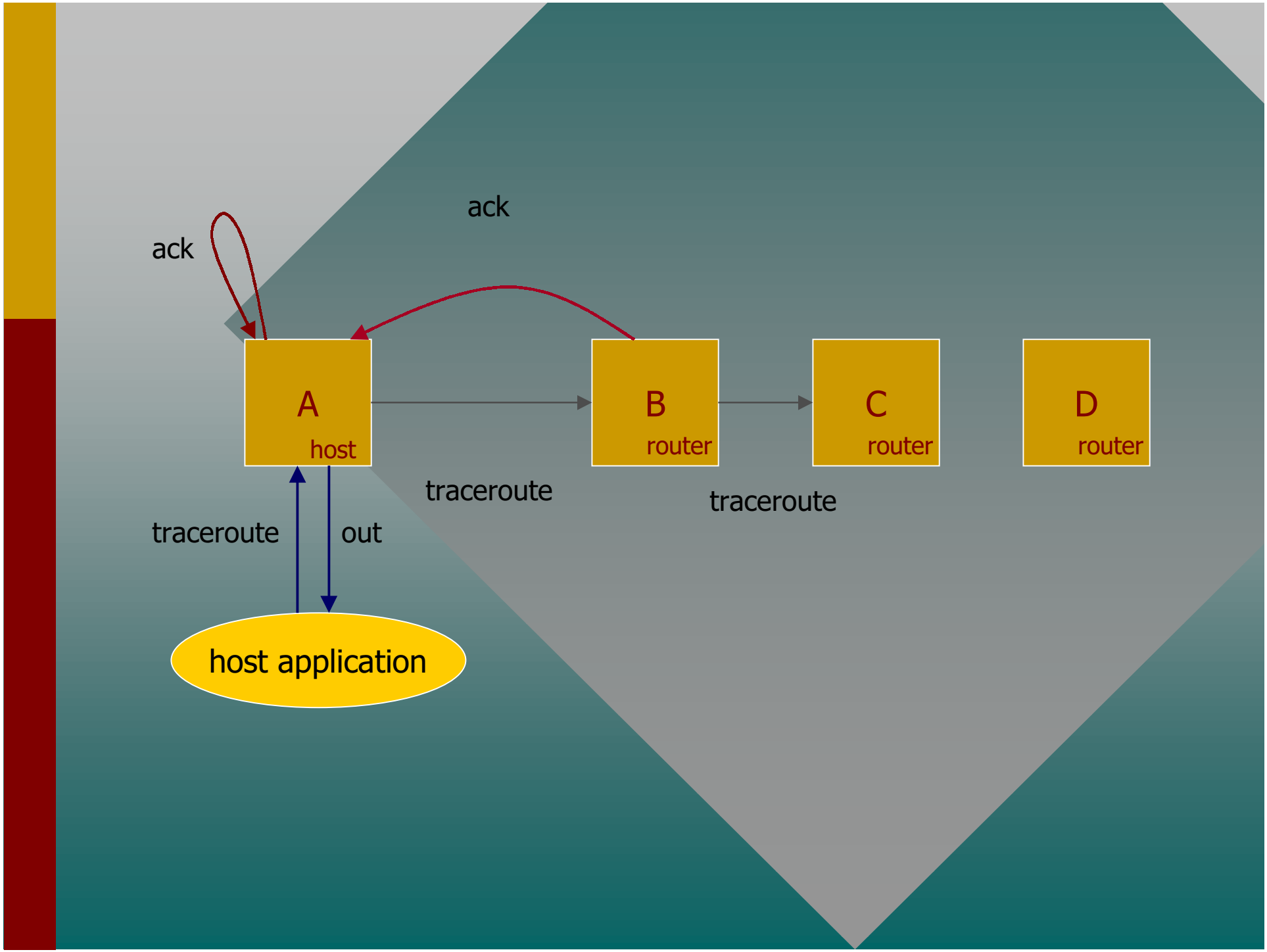
```
fun print_host(h:host, count:int) : int  =
            print(h); print(":"); print(count);  print(" " ); count+1

fun ack(l:host list): unit  =
            foldr(print_host,l,1); print("--\n"))

fun traceroute (source:host, dest:host,l:host list, count:int):unit
            let val this:host = thisHost()
            in
               (OnRemote(ack(this::l),source, count, defaultRoute);
               if (this <> dest) then
               let val  next:host = defaultRoute(dest)
                        in
                        OnNeighbor(traceroute(source, dest, this::l, count+1),
                                                next, getRB ())
                        end
               else ())
end
```

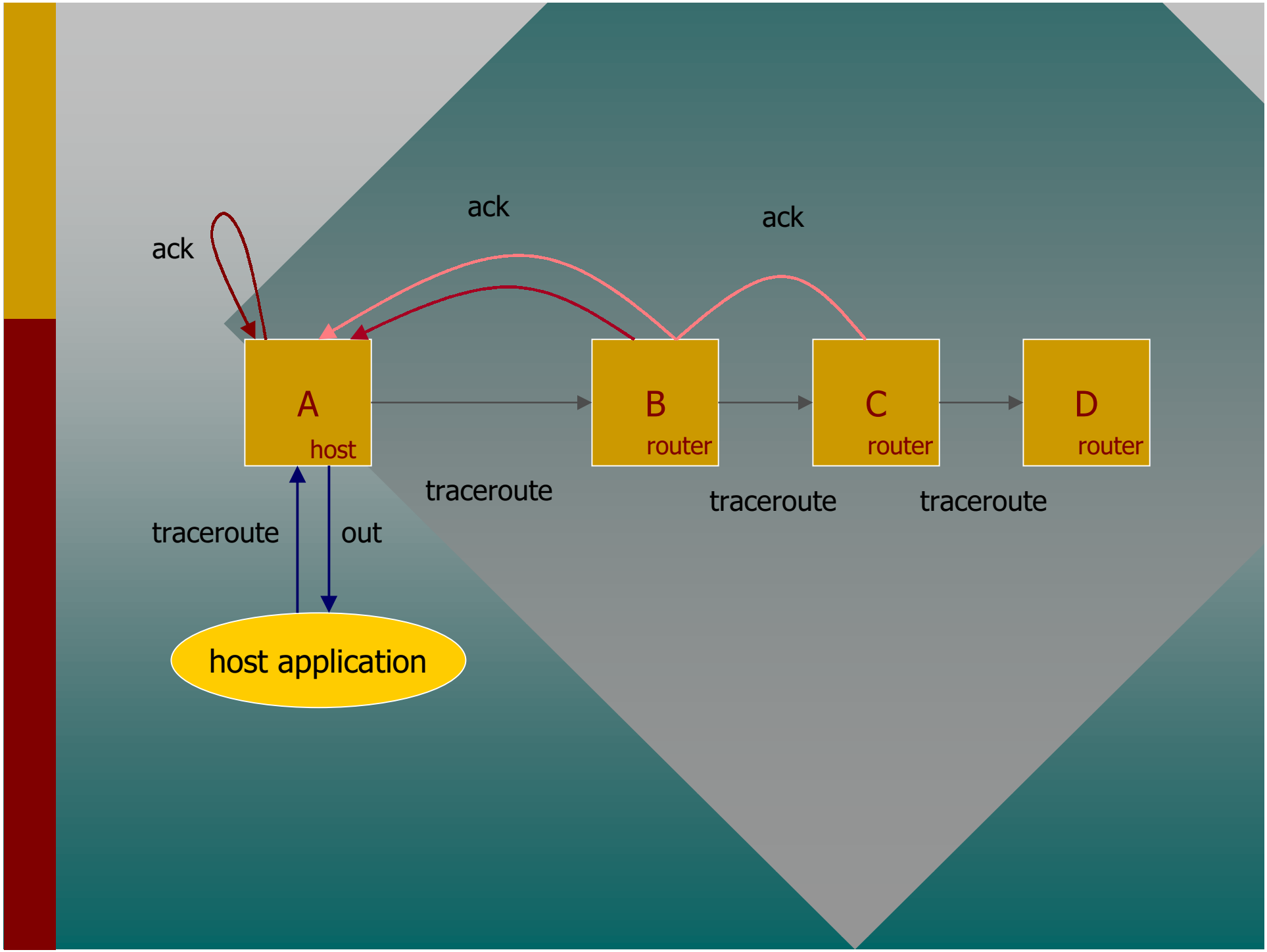
```
fun traceroute (source:host, dest:host, l:host list, count:int):unit
        let val this:host = thisHost()
        in
          (OnRemote(ack(this::l),source, count, defaultRoute);
          if (this <> dest) then
          let val  next:host = defaultRoute(dest)
               in
               OnNeighbor(traceroute(source, dest, this::l, count+1),
                                                  next, getRB, ())
                  end
          else ())
end
```

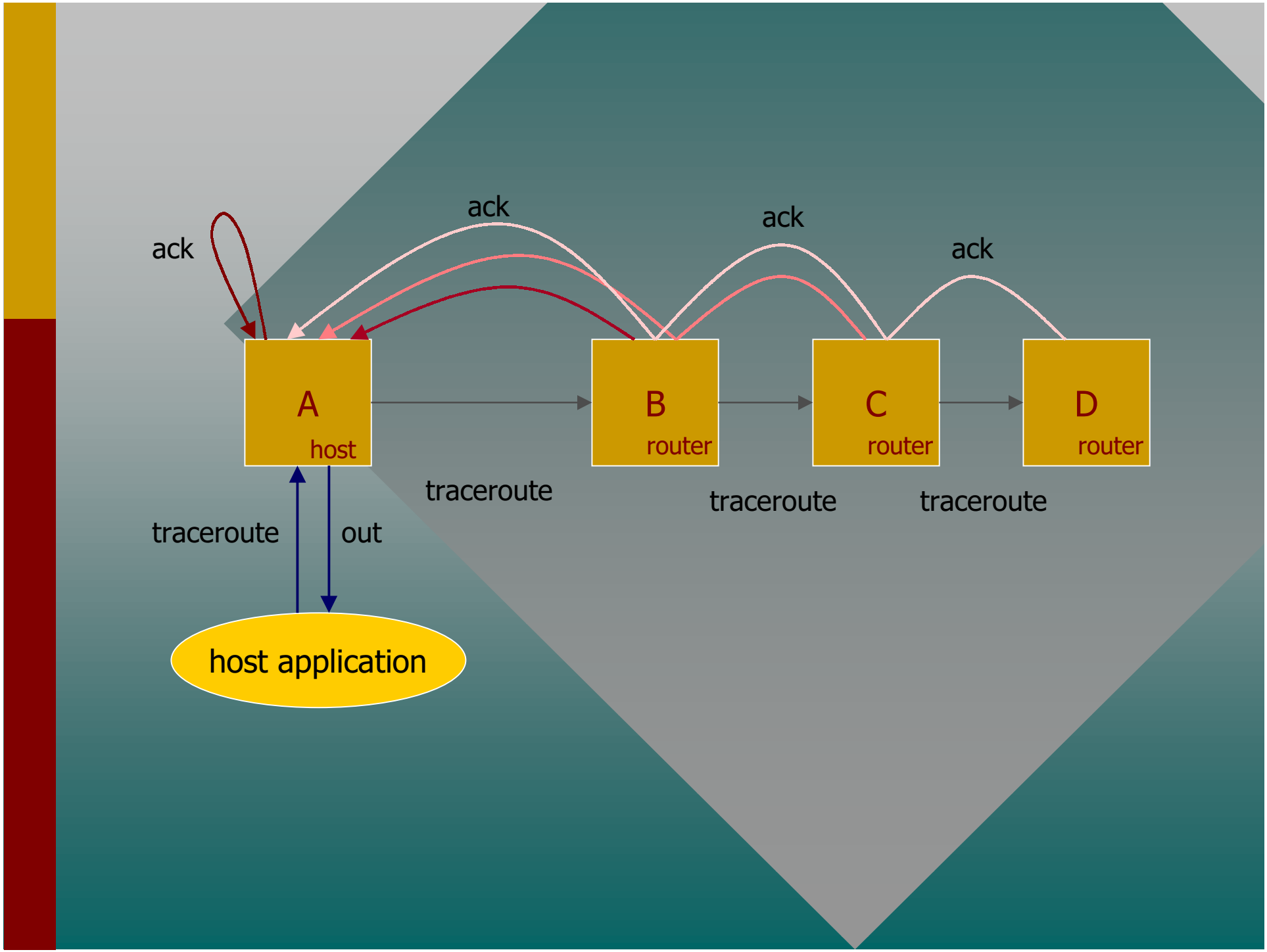
[traceroute(A,D,[],1), A, n, DefaultRoute]




ack
A
host
B
router
C
router
D
router
traceroute
traceroute
out
host application


ack
ack
A
host
B
router
C
router
D
router
traceroute
traceroute
traceroute
out
host application


ack
ack
ack
A
host
B
router
C
router
D
router
traceroute
traceroute
traceroute
traceroute
out
host application


ack
ack
ack
ack
A
host
B
router
C
router
D
router
traceroute
traceroute
traceroute
traceroute
out
host application

Delivery Program

```
fun transport(payload:blob, dest:port):unit =
    deliver (dest, payload)
```

The Service Level

- Core services (thisHost, print, defaultRoute).
- Services may change the state of a router (need for authentication).
- Example: a dynamic routing protocol.

Implementation

- Requirements:
  - dynamic code loading and
  - portability
- Implementations in
  - O'Caml, Java, Pizza
- Services are implemented in the same language as the PLAN interpreter

Implementation (ctd)

- Dynamic installation of services by passing bytecode to service installation routines
- Dynamic type checking

Related Work

- Active networks: Softnet, AIN, Active Bridge
- ANTS, Sprocket, Tacoma
- PLANET