General Autodiscovery of DTN Nodes

IETF-69

Jim Wyllie

Verizon / NASA / Ohio University

July 24, 2007

Advisors: Wesley Eddy, Shawn Ostermann, Joe Ishac

General Information

- Slide Location:
- http://irg.cs.ohiou.edu/∼jwyllie/autodiscovery-ietf.pdf
- Paper Location:
- http://www.ietf.org/internet-drafts/draft-wyllie-dtnrg-badisc-00.txt

Current State of DTN Automatic Neighbor Discovery

Generally lumped into three categories:

- Ignored
  - Assumes hosts are up (error if not)
  - Manual configuration of contact times
  - Typical for research / testing environments
- Home-brew
  - Added in a domain- and implementation-specific way
  - Typically tied to a convergence layer
- Punted
  - Assumes the lower layer can queue / figure it out
  - Can lead to bad link utilization decisions

Problems with these Approaches

- Administrative nightmare
- No interoperability: "walled" domains
- New dynamic solution implies a new protocol
- Security is nonexistent or an afterthought

So, What Do We Need?

- Generic autodiscovery mechanism (interoperability)
- Security-conscious
- Works in varying EID schemes (CBHE, etc)
- Re-use of discovery code in max. environments
- Maximum flexibility for different environments

What our Mechanism IS

- Similar to "Hello" messages of OSPF [1]
- Convergence-layer independent
- Securely identifies DTN neighbors and their capabilities
- Identifies contact windows between nodes
- Identifies contact direction

What our Mechanism is NOT

- A routing protocol
  - Though this is a step...
  - Dynamically changing neighbors with forwarding characteristics
- Find who can forward (this), and then forward to them (future work)

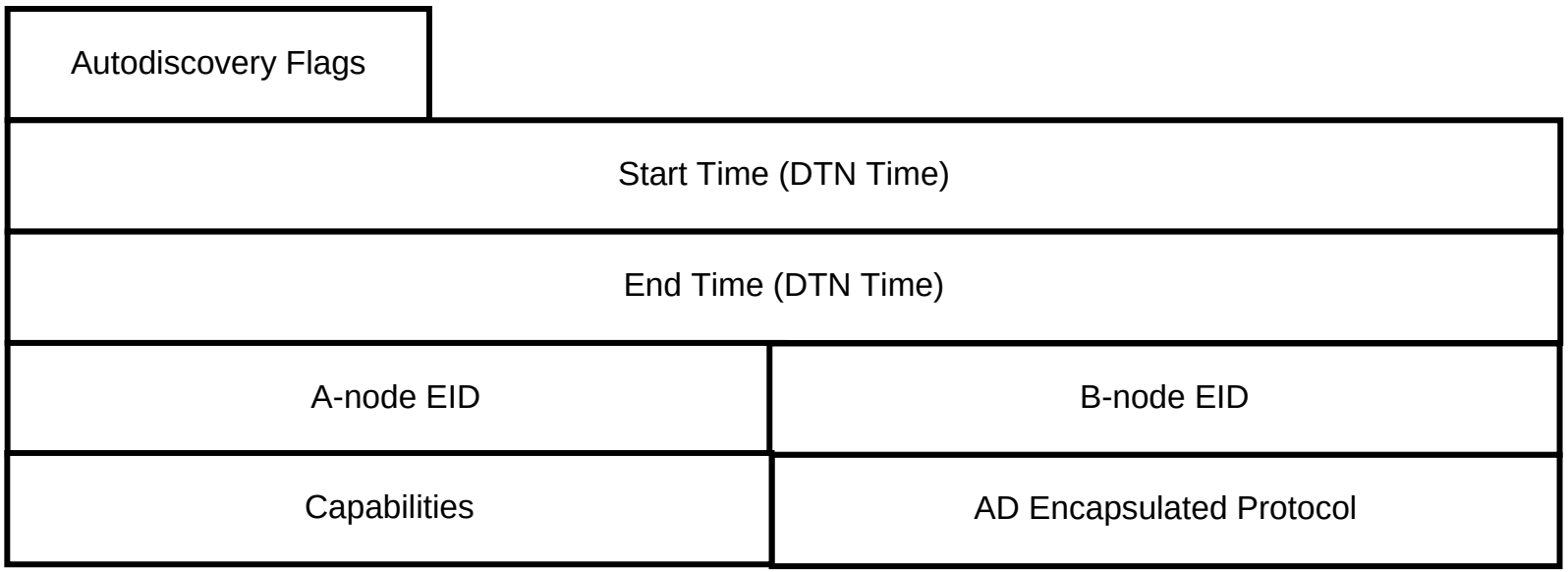
Basic terminology

- **Neighbor** – Symmetric communication over exactly one convergence layer
- **Pitcher** – Unidirectional "neighbors", only sender, no receiver
- **Catcher** – Unidirectional "neighbors", only receiver, no sender

Overall Protocol Structure

- Split into two parts
  - Domain-independent – General, required features to all autodiscovery implementations
  - Domain-specific – Custom-domain features to arrive at the independent information
- All data transmitted in the bundle payload
- A few examples to come later...

Basic Structure in Pictures



Autodiscovery Header. Approximate 'width' is 32 bits. All values (except 'encapsulated protocol') are SDNVs: sizes presented are shown for order and estimated sizes in typical deployments.

Autodiscovery Flags

SDNV flags field (SDNV implies that it can grow ’endlessly’):

- 0 – Start-time present (omit for "right now")
- 1 – End-time present (omit for "in perpetuity")
- 2 – B can send, A can receive
- 3 – A can send, B can receive
- 4 – Undefined
- 5 – Undefined
- 6 – Undefined

Times and Capabilities

- Start and End Times – DTN times
- Node EIDs – Dictionary EID references
- Capabilities
  - Willing to store / forward, speaks TCP, etc.
  - Not fully defined yet in current draft

AD Encapsulated Protocol Field

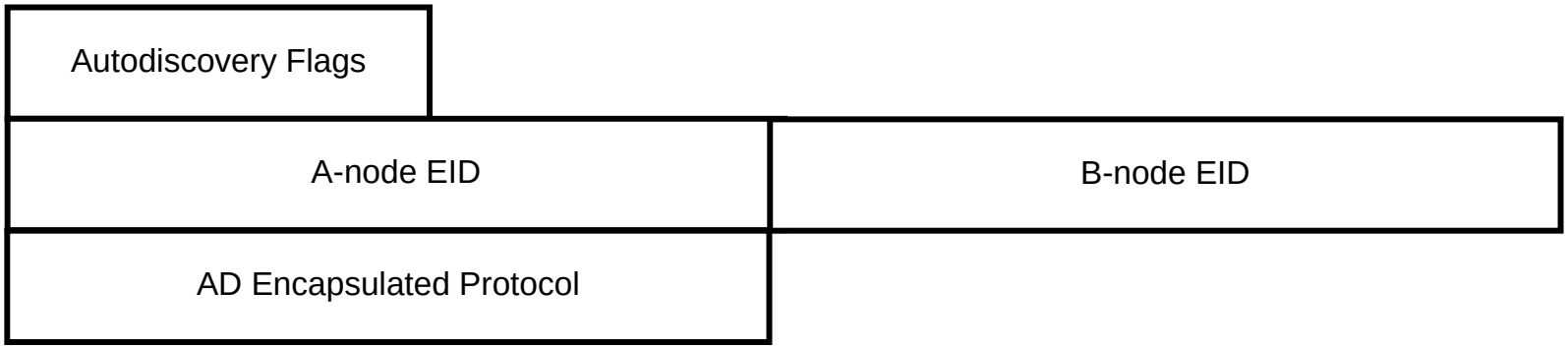
- Specifies the type of domain-specific discovery – NOT SDNV
- Protocol definitions defined separately
- 0x00 – None, no domain-specific portion
- 0x01 – DMC-like satellites
- 0x02 - 0xEF – Undefined
- 0xF0 - 0xFF – Experimental

Security Concerns

- Easy in trusted network
  - Easy to get: DTNSEC + AD = trusted management
- Can use DTNSEC itself for "public" environments

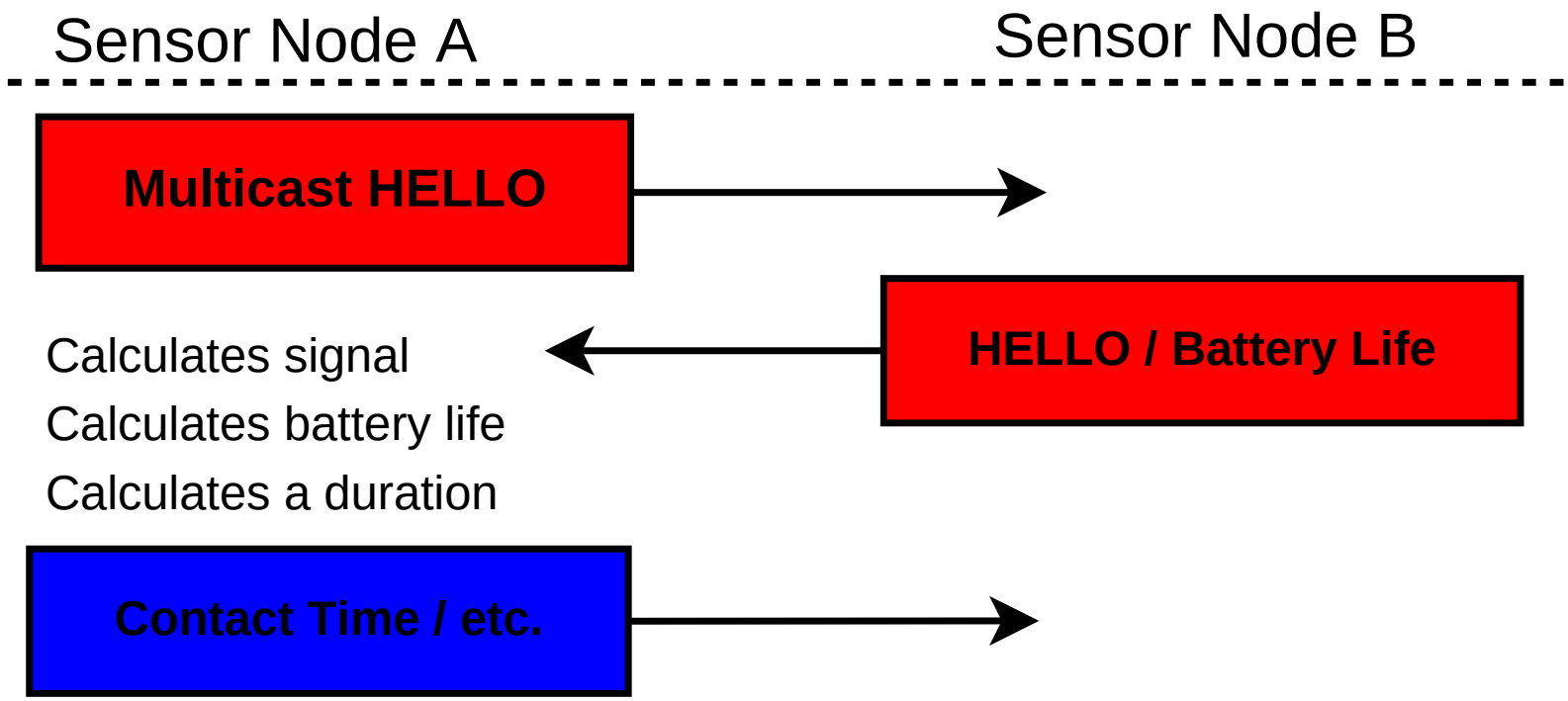
Example – Wholly domain-specific Information Header

What this header looks like in its only-domain-specific format...



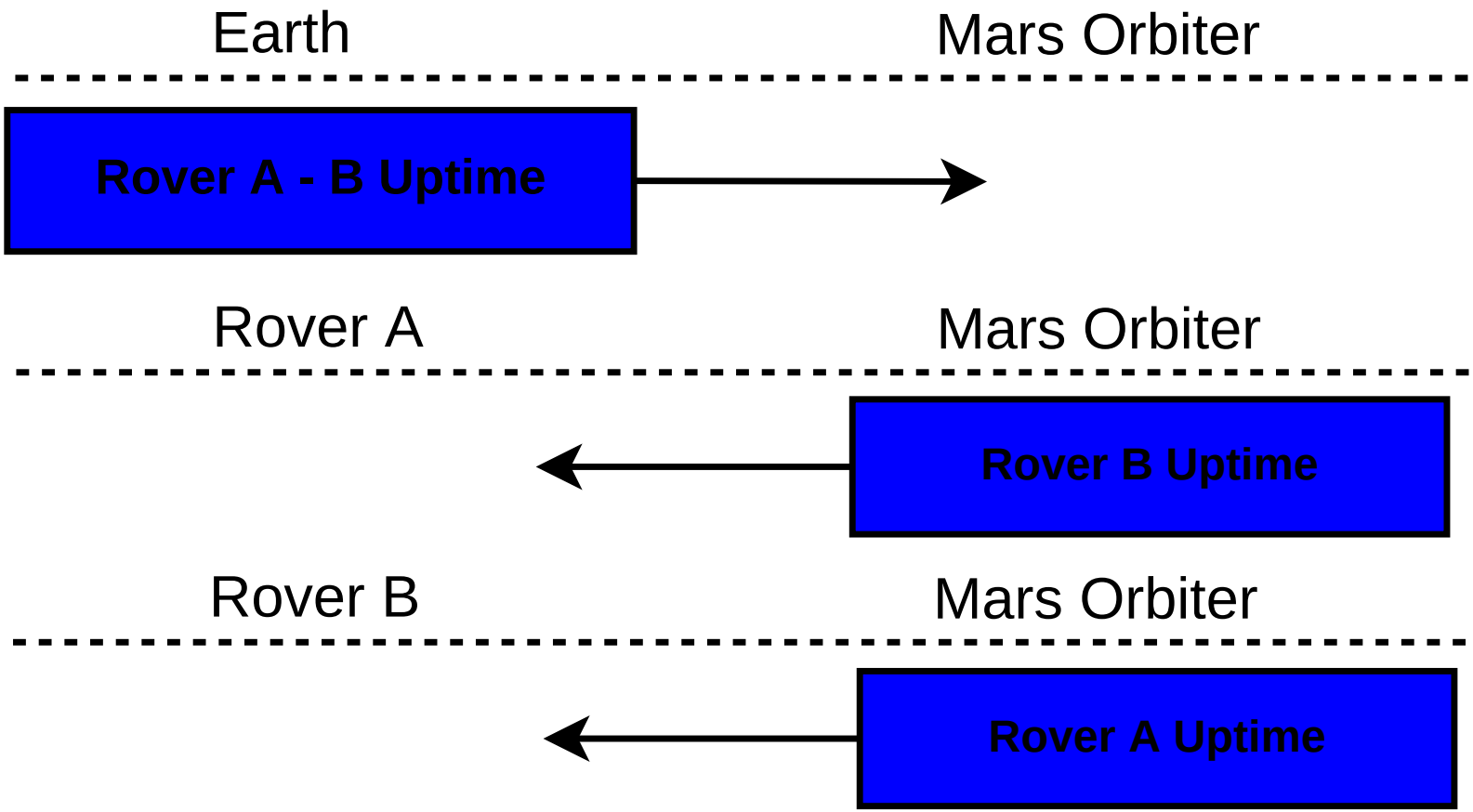
Example – Ad-Hoc Sensor Network

How two ad-hoc sensor nodes might establish contact information. Red is domain-specific, blue is domain-independent.



Example – Earth - Orbiter - Rover

Rover A was told to move in a cave: Earth has to tell Rover B that it can't communicate to Rover A.



Open Concerns

- Some domain-independent overhead in domain-specific portions
- Structuring capabilities to please everyone

Questions?

- Everyone's perfectly happy with this, right?