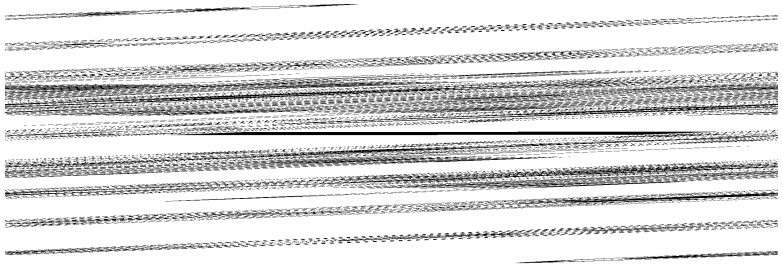
ProgDTN:
Programmable Disruption-tolerant Networking

Markus Sommer, Jonas Höchst, Artur Sterz, Alvar Penning, and Bernd Freisleben

NETYS 2022



emergenCITY SFB 1053 Natur 4.0 Sensing Biodiversity

Introduction

Disruption-tolerant Networking

- Created for space exploration
- Also useful for terrestrial applications
- Resilient against link disruption & high delays
- Store-carry-forward architecture

Terrestrial use cases

- Infrastructure is sparse
- Infrastructure has been destroyed
- Infrastructure has been degraded
- Infrastructure is overloaded
- Infrastructure lifetime is limited

Especially well-suitable for disaster scenarios

Problem

- Only limited set of DTN routing algorithms available
- They consider only logical network topology
- Network operator has to select one of the provided algorithms

What to do if no available algorithm fits your scenario?

- Deal with it
- Attempt to modify networking daemon
  - If proprietary: impossible
  - If open source: possible but cumbersome

Problem

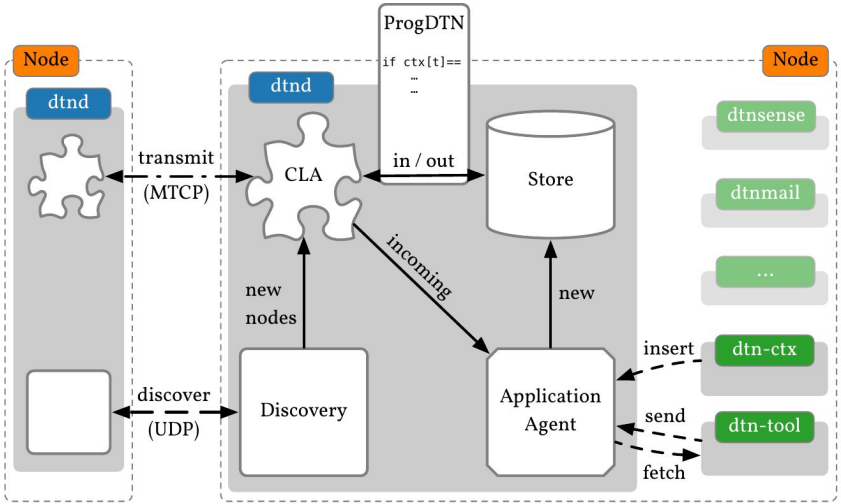
- Same issue existed in conventional TCP/IP networks
- Solved by *programmable networks* such as SDN
- Programmable networks enable optimized performance, latency, and/or throughput
- Not applicable to DTNs due to absence of central coordinator

Solution

- Make routing algorithm *programmable by the operator*
- Make *context-information* about node and neighbours available for routing decisions
- Use common, popular programming language for ease-of-use

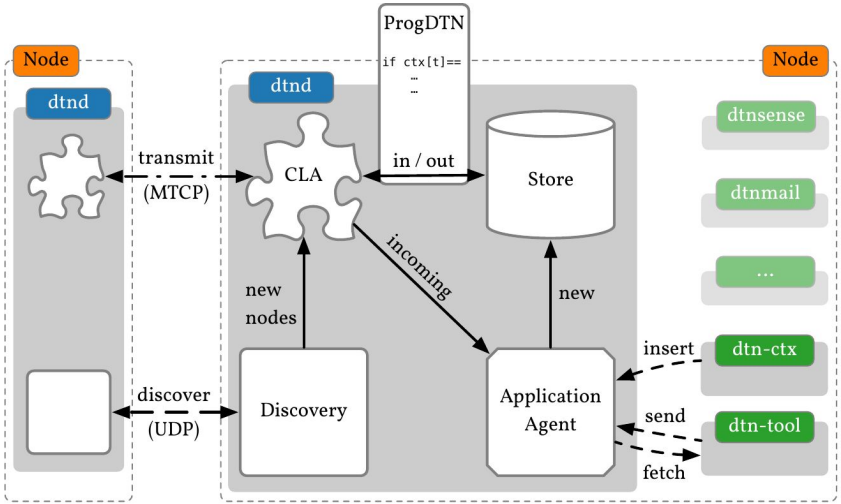
➔ This allows modification of routing behaviour without modifying DTN software
➔ Operators can use domain knowledge to design optimal algorithm tailored to their particular scenario

Architecture: DTN7 & ProgDTN



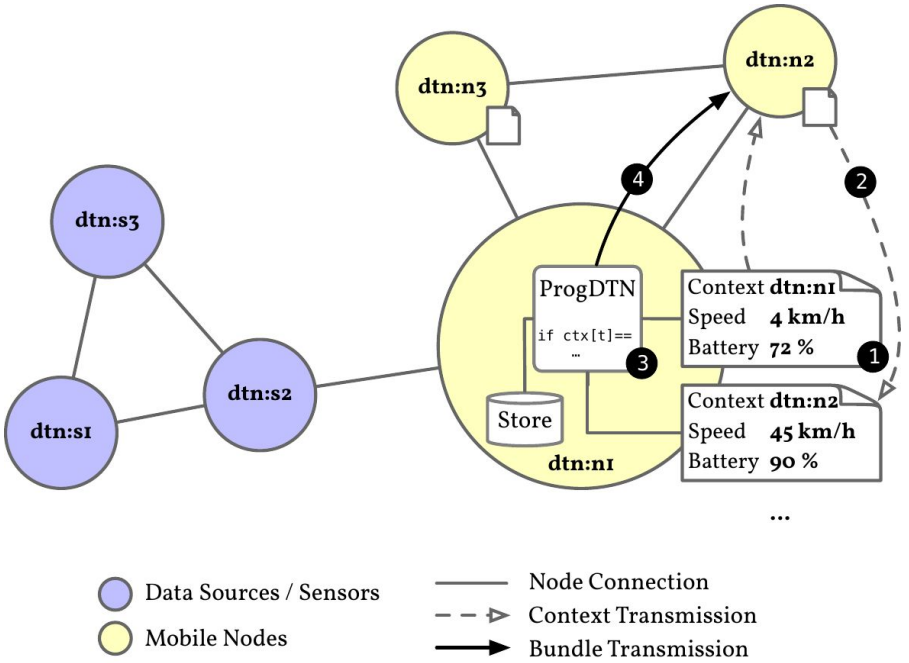
- DTN7 is a fully-featured DTN networking suite
- Implements *Bundle Protocol v7*
- Written in pure *Go*
- DTN7 provides an interface for routing algorithm implementation
- *ProgDTN* exists as such a routing-interface implementation

Architecture: DTN7 & ProgDTN



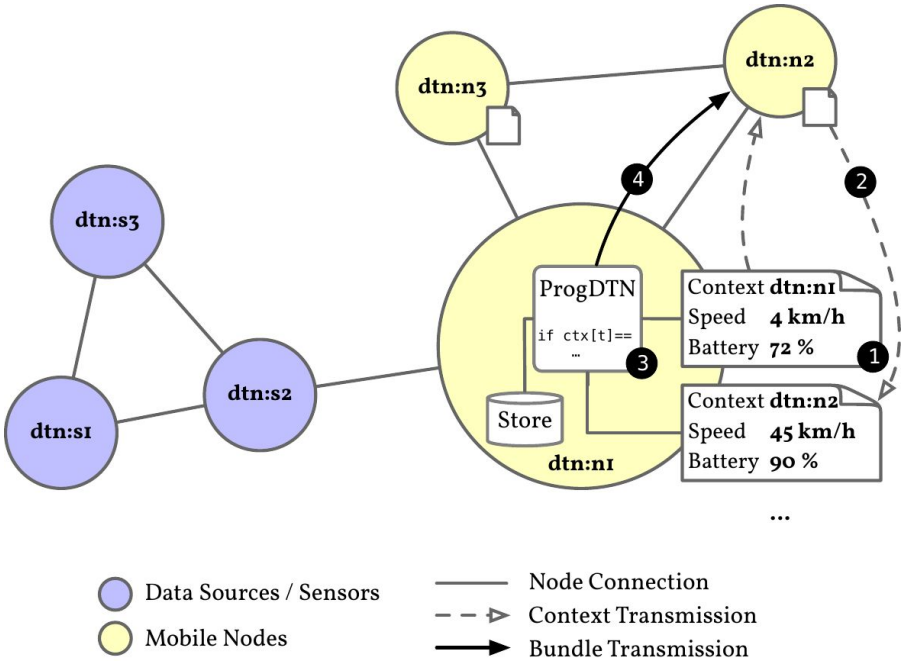
- New bundles are created via *Application Agent*
- Bundles are stored locally on-disk
- Daemon regularly dispatches bundles
- Use *Convergence Layer Adapters (CLAs)* to abstract P2P communication
- Routing algorithm selects CLAs for forwarding
- Incoming bundles are stored locally or received by application agent

Context Information & Forwarding Decisions



- Two types of context: node and bundle context
- Context can be arbitrary key-value data
- REST interface to insert context information
- Routing algorithm supplied as JavaScript-programm
- Use embedded JavaScript engine for evaluation

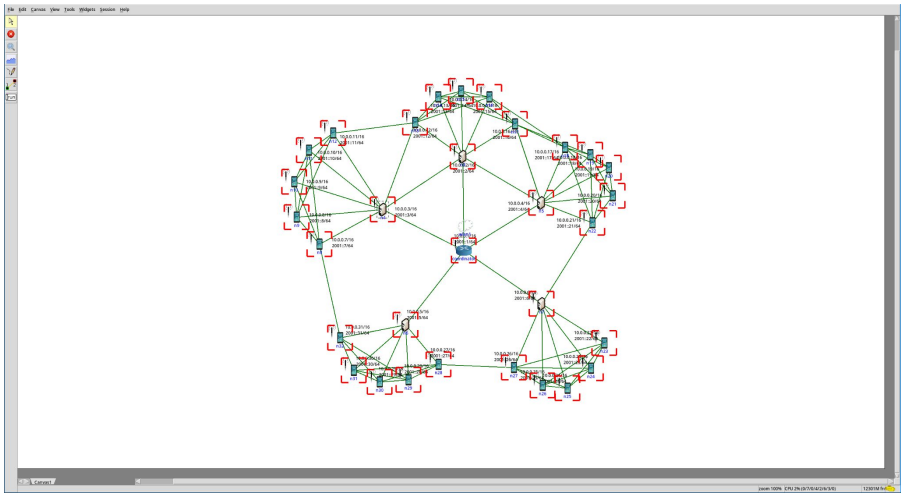
Context Information & Forwarding Decisions



1. Receive context data from external sources
2. Receive peer context information via *context bundle*
3. Evaluate own, peer, and bundle context for forwarding decision
4. Transmit bundle to peer via CLA

Evaluation Setup

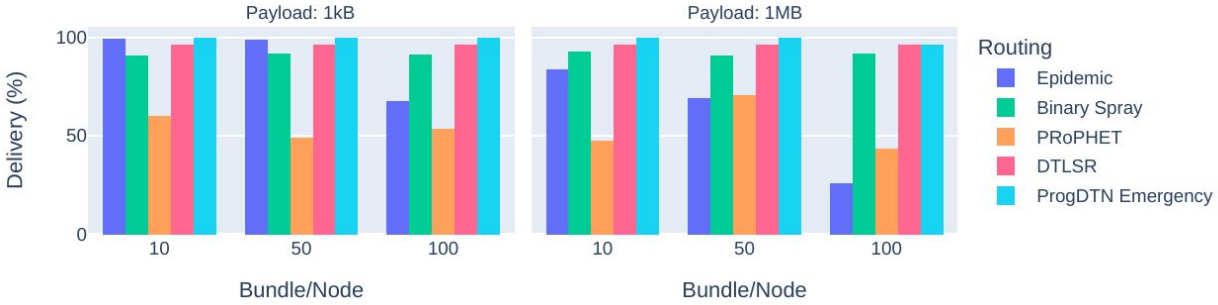
- Simulate a simplified emergency scenario
- Three node types:
  - Coordinator in the centre, sends broadcasts to all nodes
  - Responders in inner ring, relay messages between coordinator and civilians
  - Civilians in the outer ring, send emergency messages to the coordinator
- Payload sizes: 1 kB, 1 MB
- Bundles per node: 10, 50, 100
- *Epidemic routing, Binary Spray & Wait, PRoPHET, Delay-tolerant Link State Routing (DTLSR)*



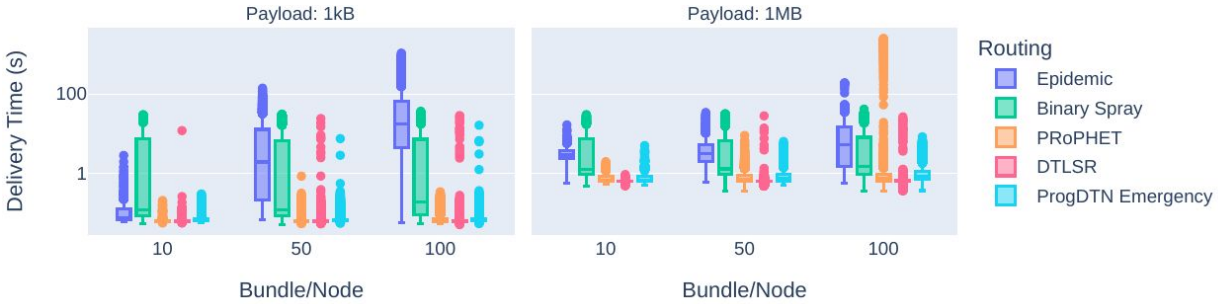
```
len = peers.length;
for (var i = 0; i < len; i++) {
    var peer = peers[i];
    var thisPeer = peerContext[peer];
    var peerRole = JSON.parse(thisPeer["role"]);
    var peerType = peerRole["node_type"];
    var forward = false;
    switch (ownType) {
        case "coordinator":
            forward = (destination === "civilian");
            break;
        case "responder":
            forward = (destination === peerType);
            break;
        case "civilian":
            switch (destination) {
                case "coordinator":
                    forward = (peerType === "coordinator" || peerType === "responder")
                    break;
                case "civilian":
                    forward = (peerType === "civilian")
                    break;
            }
            break;
    }
    if (forward) {
        senders.push(peer);
    }
}

senders;
```

Evaluation

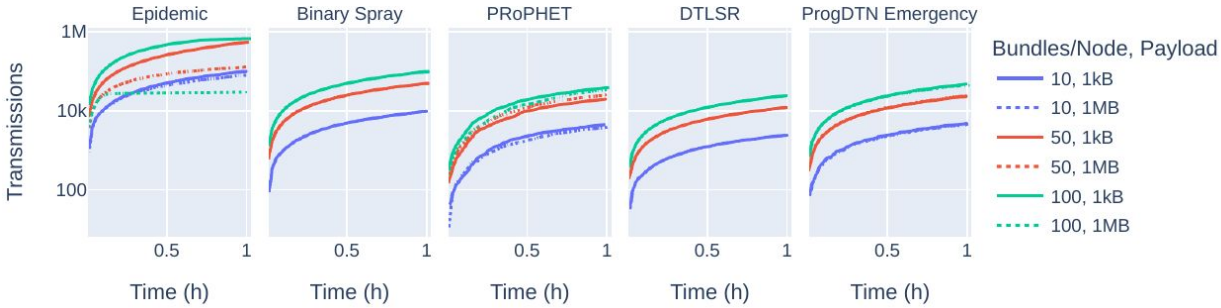


ProgDTN delivery ratio on-par or better than other algorithms

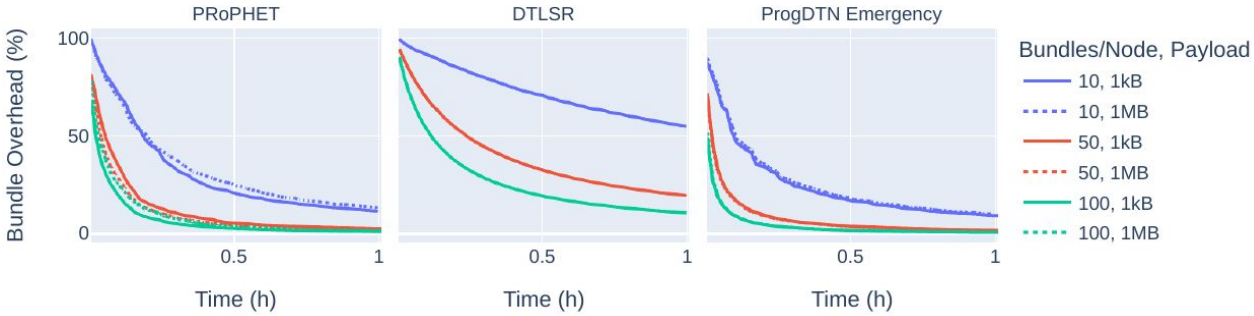


ProgDTN delivery time on-par with other algorithms, with some outliers

Evaluation

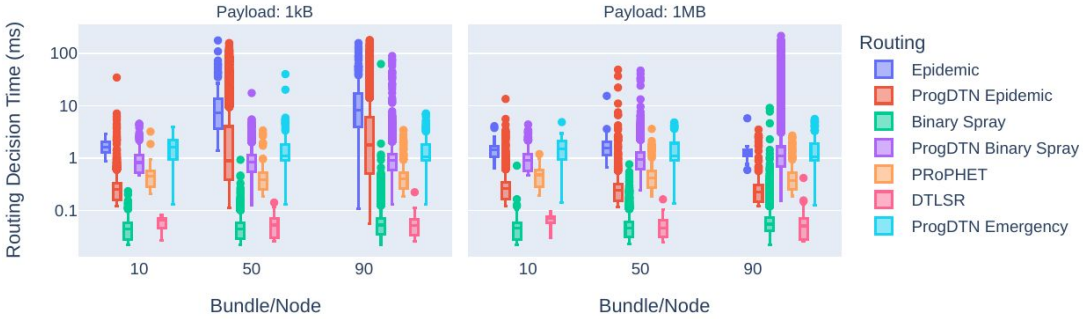


ProgDTN has far fewer transmissions than epidemic, similar to best other algorithm

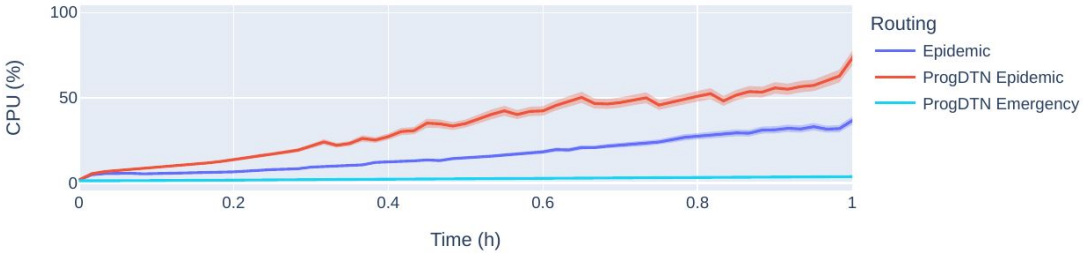


ProgDTN has similar overhead as other algorithms that send metadata

Evaluation



Making a routing decision with ProgDTN takes longer and requires more CPU cycles.



With a well designed algorithm, overall load is reduced, which saves both time and computations.

Thank you for tuning in!

Markus Sommer: msommer@inforamtik.uni-marburg.de

Jonas Höchst: hoechst@informatik.uni-marburg.de

Artur Sterz: sterz@informatik.uni-marburg.de