ProgDTN: Programmable Disruption-tolerant Networking

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

Dept. of Mathematics & Computer Science, University of Marburg, Germany
{msommer, hoechst, sterz, penning, freisleb}@informatik.uni-marburg.de

Abstract. Existing routing algorithms for disruption-tolerant networking (DTN) have two main limitations: (a) a particular DTN routing algorithm is typically designed to achieve very good performance in a specific scenario, but has limited performance in other scenarios, and (b) DTN routing algorithms do not take advantage of network programmability to profit from its benefits. We present *ProgDTN*, a novel approach to support *programmable disruption-tolerant networking* by allowing network operators to implement and adapt routing algorithms without knowledge of a router's interior workings using the popular JavaScript language. To consider the specific properties of a particular application scenario, network operators can incorporate context information of DTN bundles and nodes in their routing algorithms. ProgDTN is based on DTN7, a flexible and efficient open-source, platform-independent implementation of the Bundle Protocol version 7. Our experimental evaluation demonstrates that using ProgDTN to tailor a routing algorithm to a particular scenario achieves excellent results of up to 99.9% delivery ratio while reducing unnecessary transmissions by 92.9%. ProgDTN's implementation, our tailored scenario-specific routing algorithm, and code/data fragments for our experiments are released under permissive open-source licenses.

1 Introduction

Originating from developments related to the exploration of outer space, *disruption-tolerant networking (DTN)* has found its way into numerous terrestrial applications, e.g., in scenarios where communication networks are destroyed or disrupted and cannot be repaired for days. Apart from natural disasters, it may be hard for people to communicate via mobile devices in remote areas without a deployed telecommunication infrastructure. When end-to-end connectivity is not available, DTN can be utilized to keep communications going without the need for traditional infrastructures. However, utilizing DTN for communication requires custom network protocols, since protocols of the widely used TCP/IP stack are not well suited for such challenging situations. Therefore, several routing algorithms were developed for DTNs. Some are targeted at general purpose applications, i.e., they rely on the connectivity inside the network or are based on conventional routing schemes, such as link-state routing. Other DTN routing algorithms are constructed for specific scenarios, such as the movement of

people during a workday or emergency situations. We argue that each DTN routing algorithm is designed for a specific scenario in which it achieves very good performance, but has limited performance in other scenarios.

Conventional network protocols of the TCP/IP stack suffer from the same limitations, which motivated the introduction of *programmable networks* in the literature. For example, Software-defined Networking (SDN) offers programmable means of configuring networks using customized algorithms for processing and routing. Typically, this leads to optimized performance in terms of latency, throughput, and/or resilience. We argue that DTN should also take advantage of network programmability to profit from its benefits like time and cost savings, reduction of human error, customization, and improved performance. However, the SDN approach cannot directly be transferred to DTN, because SDN, in general, requires a coordinating entity that deploys (programmable) rules to network nodes. In most cases, a coordinating entity is not present in DTNs.

In this paper, we present *ProgDTN*, a novel approach to support *programmable disruption-tolerant networking* by allowing network operators to program a node's routing behavior based on DTN bundle metadata, additional bundle context, and node context. Node and bundle context can be used to reflect the specifics of a particular scenario, e.g., the speed of a node in a mobile scenario, the battery level of nodes in a scenario without fixed power supply, or geographic information of the context of a bundle. ProgDTN consists of a programming interface that allows a network operator to program the routing algorithm without knowledge of the router's interior workings. ProgDTN's implementation is based on DTN7 [16], a flexible and efficient open-source, platform-independent implementation of the Bundle Protocol version 7 [6], and uses JavaScript as the programming language, because it is widely used and easy to understand. In our experimental evaluation, we compare ProgDTN to four existing routing algorithms. We demonstrate that a programmable DTN routing algorithm tailored to a specific scenario achieves excellent results in terms of up to 99.9% delivery ratio while reducing unnecessary transmissions by 92.9% compared to other state-of-the-art DTN routing algorithms in an emergency response scenario. We achieve a low delivery time of bundles (1 – 15 seconds) and a low overhead in terms of CPU utilization and routing decisions. Our contributions are:

- We present ProgDTN, a novel approach for programmable DTN routing.
- We show that using ProgDTN, network operators can tailor routing algorithms to their individual scenarios by incorporating context information.
- We present a comparative experimental evaluation of ProgDTN, achieving excellent results in terms of delivery ratio, delivery times, and overhead.
- We make ProgDTN's implementation, our scenario-specific routing algorithm,[1] and code/data fragments of our experimental evaluation[2][3] available under permissive open-source licenses.

[1] `https://github.com/umr-ds/dtn7-go/tree/progdtn`
[2] `https://github.com/umr-ds/progdtn-evaluation`
[3] `https://dshare.mathematik.uni-marburg.de/index.php/s/8k6XZgKJp9kTMPS`

The paper is organized as follows. In Section 2, an overview of related work in the field of delay-tolerant routing is given. Section 3 and 4 cover ProgDTN's design and implementation. Section 5 discusses the results of our experiments. Finally, Section 6 concludes the work and outlines areas for future work.

2 Related Work

This section gives a brief overview of related work on context-aware routing in general and context-aware routing in DTN.

Context-aware Routing. Apart from using information about the network topology, context-aware routing algorithms use additional information to make routing decisions. Here, we focus on context-aware routing protocols for wireless mobile ad hoc networks (MANETs).

The History-based Routing Protocol for Opportunistic Networks (HiBOp) uses social information, such as club memberships or home addresses to infer the likelihood of encounters and improve routing decisions [5]. Dynamic Social Grouping-based Routing (DSR) is a routing algorithm that harnesses social grouping for efficient routing in ad hoc networks [7]. The Inheritance Inspired Context-Aware Routing Protocol (IICAR) follows a biology-inspired approach to routing based on Mendel's laws of inheritance [3].

The protocol proposed by Biswas et al. [4] uses the properties of ad hoc wireless networks. It maintains static context, such as node and interface types and social information, and dynamic context, e.g., geolocation, channel quality, and encounter frequency. The information is used to calculate utility scores, combined using a routing metric to determine a delivery probability for each message. Messages are forwarded to a) the closest short-range neighbor and b) a long-range neighbor selected based on the delivery probability and the distance.

The algorithm proposed by Errouidi et al. [9] uses context information to improve resilience in MANETs. It employs a fuzzy logic system for three context metrics, a node's remaining energy storage, distance between peers, and node mobility, to judge its "stability". This allows it to improve routing decisions by avoiding unstable nodes, which improves network stability and delivery metrics.

Context-aware Delay-tolerant Routing. The Context-Aware Adaptive Routing (CAR) protocol harnesses context information for routing decisions DTN and combines synchronous and asynchronous transmission [14, 15]. Messages are transmitted synchronously using a distance vector routing approach if the recipient is located in the same neighborhood. If no direct connection between sender and recipient exists, asynchronous (DTN) transfer is used, where a node computes delivery probabilities of the directly connected nodes based on its context information. It is up to the network provider to define concrete attributes, utility functions, and weights for the generic approach.

The Sensor Context-Aware Routing (SCAR) protocol for distributed sensor networks [13] is based on the CAR protocol, omitting the distinction between

synchronous and asynchronous transmissions. The approach only handles a single specific use case and does not attempt to provide a general routing algorithm.

A further routing scheme that competes with CAR was proposed by Johari et al. as Context-Aware Community Based Routing (CACBR) [11]. Message forwarding is based on a combination of network and context parameters, such as communities a peer belongs to, message delivery and forwarding history, and available buffer space and battery level.

In general, schemes that attempt to harness social and community relations are relatively common; another example is the Socially-Aware Adaptive Delay Tolerant Network (DTN) routing protocol by Ullah and Qayyum [19]. It utilizes a metric called degree centrality to estimate how well a node is embedded in a community to drive forwarding decisions.

Beak et al. [2] propose a version of the PRoPHET routing algorithm, improved by using context information. While the authors show that their proposed protocol outperforms regular PRoPHET, it still relies significantly on the underlying protocol, and the use of context information is limited.

Another approach was proposed by Rosas et al. [17]. It is not focused on designing a routing algorithm that includes context information, but instead measures the performance of different algorithms under different context values and then uses future context information to choose the optimal one.

To the best of our knowledge, no attempt has been made in related work to create a general-purpose, context-aware, programmable DTN routing system.

3 ProgDTN Design

In this section, we present the design of ProgDTN, including DTN fundamentals, system requirements, context information, and ProgDTN's architecture.

3.1 DTN Fundamentals

The data transmission unit of the *bundle protocol* is a *bundle*, which consists of multiple *blocks*. Each bundle must contain a *primary block* containing basic metadata, such as the bundle's ID, sender and recipient IDs, and a *payload block* that carries the bundle's payload. A bundle can also include an arbitrary number of *extension blocks*. While the DTN standard specifies several extension block types, it allows implementations to specify additional types.

A node in DTN operates in a store-carry-forward manner, i.e., when a bundle is received, it is stored in local, long-term storage, from where it will be regularly forwarded. When forwarding, the network daemon invokes the configured routing algorithm to select a subset of currently connected peers to which the bundle should be forwarded. The actual peer-to-peer connection is abstracted in a so-called *convergence layer* (CL) that may use any lower-layer communication protocol to achieve data transmission. ProgDTN is designed to be entirely independent of any specific communication infrastructure and works with any standard-compliant convergence layer.

3.2 System Requirements

ProgDTN's goal is to allow network operators to develop scenario-specific routing algorithms without modifying the DTN software itself. Furthermore, changes to the routing algorithm should not require recompilation of the DTN software to reduce the complexity of deploying new or adjusted routing algorithms. Instead, the forwarding rules are loaded at startup from a provided script file and interpreted by the DTN software. In this way, algorithms may be swapped by restarting the DTN software and giving it a different file to load. To specify these forwarding rules, we use a general-purpose programming language and embed an interpreter into the DTN software. This allows maximum flexibility without having to learn a new domain-specific language.

3.3 Context Information

ProgDTN belongs to the class of context-aware routing algorithms, i.e., routing decisions may be based on additional information of the environment in which the network exists. *Context* refers to any information about the nodes, bundles they are transmitting, or any other information that the network operator may deem helpful. ProgDTN does not put any semantic restrictions on context information, except that each piece of information needs to be uniquely named. Instead, we provide network operators with a generic, powerful interface for generating and processing relevant context information.

Context Types. ProgDTN distinguishes between two classes of context information, *bundle context* and *node context*. Bundle context is any additional information attached to an otherwise normal bundle, e.g., the physical location of the bundle's recipient, information about the originator or the recipient, and really anything that the network operator might think of. On the other hand, node context is information about a specific node, such as the node's location, the node's battery status, and its connectivity status. This information is not usually attached to other bundles, but if it must be communicated to other nodes, the node broadcasts it using a special context bundle.

Context Generation. The naive approach for generating context information would be to have the DTN software itself generate the necessary information. This would, however, violate the ease of use goal, since it would require modification of the DTN software's code for each scenario. ProgDTN adopts an approach where context generation is left up to external programs. For this purpose, the DTN software exposes an interface through which context information can be injected. This interface should be based on a widely used communication protocol/architecture to ensure ease of use.

Context Transmission. Since context information needs to be attached to a bundle, and since it may be helpful for nodes to be able to exchange their contexts, we defined a custom extension block to carry context information.

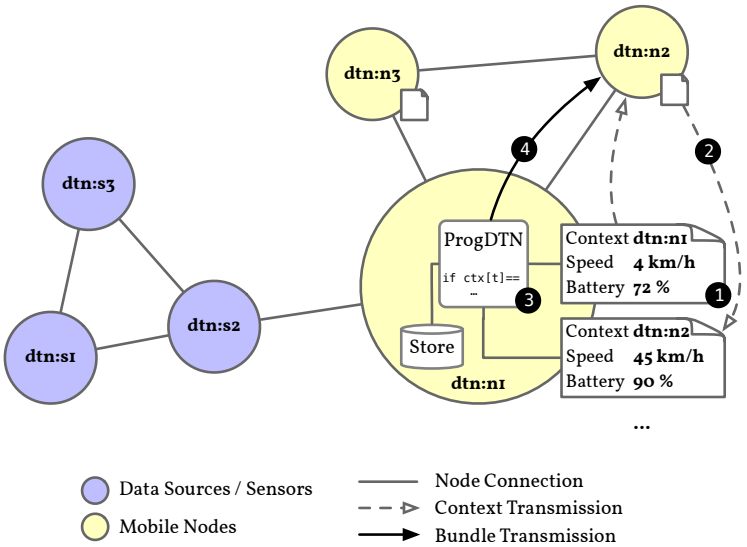

Fig. 1: Architectural overview of a DTN deployment utilizing ProgDTN

This extension block may either be attached to an existing bundle or be used to exchange context data with peers by sending a special *context bundle*. Extending a regular bundle with a context block allows nodes to use this information when making forwarding decisions.

3.4 ProgDTN Architecture

Fig. 1 shows the architecture of a DTN deployment utilizing ProgDTN. The yellow circles depict mobile DTN nodes, whereas the violet circles show data sources in a hypothetical scenario. Each node is identified by the prefix `dtn:` followed by a name, such as `n1`, used as the address for bundle transmissions. The lines between the circles show connections between the particular nodes. Node `dtn:n1` is used to visualize further concepts.

In general, when a bundle enters a DTN node, either by being created or received from a peer, it is placed into the node's local on-disk storage (`store` in Fig. 1). The bundle is then passed to the configured routing algorithm to select peers for forwarding. ProgDTN takes the position of the routing algorithm, which is visualized by the box entitled `ProgDTN`. It takes the particular bundle, which can itself have attached context information, and the context information for its node (shown as ❶ in Fig. 1), as well as the peer's context (❷). The context in this example is the speed and battery level of the transmitting node and for node `dtn:n2`. The interpreter (❸) then executes the routing script provided by the network operator, which must be aware of the available context data, and filters the list of connected peers to select the subset for forwarding. Once the routing

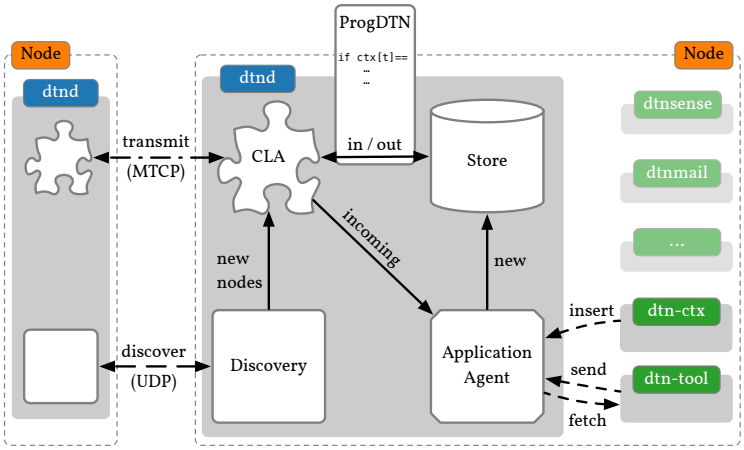

Fig. 2: `dtn7-go` with the ProgDTN implementation between CLA and Store

program has returned the list of selected peers, the DTN software transmits the bundle to the selected peers (❹).

4 ProgDTN Implementation

ProgDTN's implementation is based on `dtn7-go`, a powerful DTN software suite developed in the DTN7 project[4]. `dtn7-go` implements the most recent draft of the bundle protocol (BP) version 7 [6] in the GO programming language [16].

Fig. 2 shows the components of `dtn7-go`. The two dotted boxes indicate two DTN nodes, and the arrows between them show their interaction, data transmission (`transmit`), and peer discovery (`discover`). The arrows within the `dtnd` box visualize the data flow of bundles within a node. The right-hand side shows multiple tools that use an application agent (AA) to insert data to or retrieve data from `dtn7-go`. The AA then stores the received data in the node's local store. The bundle protocol abstracts peer-to-peer communication using convergence layer adapters (CLAs), which can use a variety of protocols and technologies such as TCP, UDP, or even e-mail sent over Bluetooth. Each CLA exposes a defined API to the daemon, which can then transparently send & receive messages without having to bother with data serialization or transmission. To support connections in dynamic networks, `dtn7-go` uses a peer discovery mechanism that continuously broadcasts information on all of the node's CLAs and notifies the daemon about newly discovered peers. For forwarding decisions, `dtn7-go` includes an API that can be used to implement routing algorithms. Any *Go* datatype that satisfies this interface can be used as the router, and `dtn7-go` ships with various established routing algorithms. ProgDTN consists of an im-

[4] `https://dtn7.github.io/`

plementation of that interface and a set of extensions for receiving and storing bundle and node context.

4.1 Using JavaScript for Programmable Routing

We decided to use JavaScript as our general-purpose programming language for programmable DTN routing due to the following reasons: (a) JavaScript is very popular; according to a 2021 survey conducted by Stack Overflow[5], nearly 65% of developers work in JavaScript, giving it a roughly 17% lead over the second-placed general-purpose language Python at approximately 48%; and (b) JavaScript can be embedded into the GO programming language; there are several JavaScript interpreters written in GO, e.g., `goja`[6], which implements the ECMAScript standard version 5.1 with some additional features.

Thus, a routing algorithm is specified in a JavaScript file that gets loaded during startup of `dtn7-go` and compiled to bytecode representation for faster execution. Whenever a routing decision is made, the compiled JavaScript routing algorithm is invoked, which invokes a virtual machine (VM) able to interpret the JavaScript bytecode.

4.2 Programmable Routing Decisions

Any custom routing algorithm has to comply with the following API. First, the JavaScript code receives the bundle itself passed as a JavaScript object [7], as well as the bundle ID, represented as a string. Second, the ID of the bundle's source (as a string) and a list of strings representing the IDs of all currently available peers are passed to the routing algorithm. The final pieces of information are the bundle context, the node's context, and the context of all available peers. Whenever the algorithm receives context data, this data is encoded as JSON, a data serialization format that works well with JavaScript. The bundle's context can be updated using a callback function whenever this may be necessary. If logging is required, it is possible to use the passed `loggingFunc` function to write an arbitrary string to the `dtn7-go`'s logs. The algorithm may then use any or all of this data to perform the actual forwarding decision. Any JavaScript code can be executed here, including third-party libraries. ProgDTN does not place any restrictions on the possible context data; it is up to the network operator to be aware of runtime requirements. Finally, the routing algorithm must return a list of node IDs to which the bundle should be forwarded. `dtn7-go` then takes care of forwarding the bundles to the chosen nodes using the corresponding CLAs.

[5] `https://insights.stackoverflow.com/survey/2021`

[6] `https://github.com/dop251/goja`

[7] The description of the bundle data structure is omitted for brevity. We refer to `https://github.com/dtn7/dtn7-go/blob/d3b5e62a7f89994ececf98978bae499f32cc920f/pkg/bpv7/bundle.go` for further information.

4.3 Providing Context

To provide context information about the local node, we implemented a REST interface that receives information formatted in a key-value manner, where the value contains arbitrary data formatted as JSON. The context information is then saved in a global dictionary so that newer data for a given peer overwrites existing data. While the local node's context information is vital for routing decisions, so is its peers' context. Therefore, whenever two peers connect, they exchange their context information. Finally, for bundle context, the provided REST interface for bundle submission is extended to add context information to the bundle during its creation, again as arbitrary key-value pairs.

5 Experimental Evaluation

5.1 Emulation Environment

Network Emulation. To perform a large number of experiments, we used the *Common Open Research Emulator (CORE)* [1], an open-source network emulator[8]. CORE supports the execution of native binaries without re-implementing protocols, i.e, real-world code can be executed. Furthermore, we used the MACI experimental orchestration framework [10] to schedule a large number of experiments. All experiments were executed on an AMD EPYC 7742 server with 128 physical cores and 1 TB RAM, which executed up to 3 experiments in parallel.

Network Topology. We use a network topology that simulates a disaster scenario involving three parties, *civilians*, *responders*, and a *coordinator*. The scenario consists of 31 nodes arranged in a 2-circle topology, with the singular coordinator located in the center, five responders arranged in a circle around the coordinator, and 25 civilians arranged in the outer circle. Each responder is connected to 5 civilians, and the civilian clusters are connected on their edges. Each civilian sends bundles addressed to the coordinator, simulating information moving up the chain of command. The coordinator produces broadcast bundles that are supposed to be received by all civilians, which simulates announcements by the authorities to the population. The experiments simulate a network with a bandwidth of 54 MBit/s, 20 ms of delay, and a range of about 40 meters.

Experimental Parameters. All experiments are uniquely defined by a set of four parameters, summarized in Table 1 and discussed below. In total, 210 experiments were executed for one hour each. *Bundles per node* determine how many bundles each civilian and the coordinator send to the network. To avoid every node sending its data simultaneously, bundles are sent at (uniformly distributed) random times throughout the experiment. We used two different *payload sizes*, 1 kB and 1 MB, to mimic different use cases, with 1 kB serving as a stand-in

[8] `https://coreemu.github.io`

Table 1: Evaluation Parameters

Parameter	Values
Bundles per Node	10, 50, 100
Payload Size	1 kB, 1 MB
Routing Algorithm	Epidemic Routing, Binary Spray & Wait, DTLSR, PRoPHET, ProgDTN Epidemic, ProgDTN Binary Spray & Wait, ProgDTN

for text messages and 1 MB being around the size of a small image. One of the two payload sizes was used for all nodes during every experiment. To maintain reproducibility and to reduce the chance of unfavorable initial conditions, each experiment was re-run five times with pre-determined PRNG-seeds.

Seven routing algorithms are used for comparison: *Epidemic Routing* [20] is the most widely used routing algorithm in DTN, sending bundles to all peers.

Binary Spray & Wait [18] is a modified version of Spray and Wait. A node holds n copies of a bundle, half of which are transferred to the first peer. The second peer then receives half of the remaining copies and so on. Each node that receives multiple copies of a bundle proceeds in the same way. A node will only forward the bundle to its intended recipient when only a single copy remains. Spyropoulos et al. [18] suggest a value of 10 as a reasonable initial multiplicity, which we adopted in our experiments.

PRoPHET [12] exploits the fact that in DTNs, nodes usually encounter each other more than once. Whenever two peers connect, they compute a probability of meeting again in the future. This probability declines over time if these particular nodes do not meet again. A node will forward a bundle to another node if the receiving node's probability of meeting the bundle's recipient is higher than the forwarding node's probability. We used the same parameters as the authors in their original paper for calculating delivery probabilities.

Delay-Tolerant Link-State Routing (DTLSR) [8] other than classical link-state routing once a link is lost, it is not immediately removed from routing considerations, but rather "tagged" with the time since the disconnection. When the routing table is computed, this time is interpreted as a link cost of Dijkstra's algorithm to find the shortest path between the current node and the destination.

ProgDTN Epidemic / Binary Spray & Wait are re-implementations of the respective algorithms in ProgDTN, which serve primarily to compare computational overheads. All parameters are the same as in the native implementations.

ProgDTN Emergency is a custom algorithm implemented using ProgDTN and tailored to the given scenario of our evaluation, where data only flows in two "directions": from civilians to the coordinator, or vice versa. Whether a node will forward a bundle to another node depends on three factors: (a) node type (coordinator, responder, civilian), provided at startup, (b) peer type, received by a node via a context bundle and (c) bundle type (unicast to coordinator, or broadcast to all civilians), carried by a bundle in a context block attached at

bundle generation time. Unicasts (i.e., bundles from the civilians to the coordinator) are only forwarded along the inward direction, from civilians to responders to the coordinator, while civilians do not send their bundles to each other. The same applies to responders. Broadcasts flow outward, i.e., from the coordinator to the responders to the civilians; civilians distribute messages among each other. In both cases, responders do not forward bundles among other responders, but only serve as relays between civilians and the coordinator.

5.2 Results

We consider six metrics divided into two categories: network utilization and an overhead analysis. The network utilization metrics are the percentage of bundles successfully delivered, the duration of delivery, and the load generated in the network. Our overhead analysis considers the time to decide to whom a bundle should be forwarded, the percentage of bundles that do not carry a payload (metadata or context bundles), and how heavily a node's CPU is utilized.

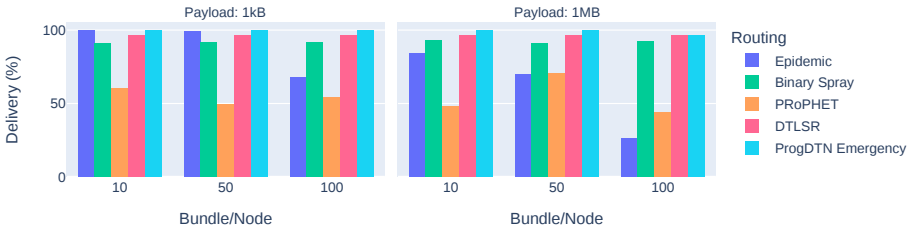

Fig. 3: Ratio of successfully delivered bundles for different parameters

Delivery Ratios. Fig. 3 shows the *delivery ratio*, i.e., the percentage of sent bundles that reach their destination. Each group on the x-axis represents a set of bundles per node, the y-axis shows the reached percentage of delivered bundles, the color denotes the routing algorithm, and each sub-figure shows the different payload sizes. For unicasts, successful delivery means that the coordinator receives the bundle, while for broadcasts, all potential recipients (i.e., the civilians) need to receive the bundle. The performance of ProgDTN Emergency is at least equal to all other routing algorithms. In many cases, it outperforms the other routing algorithms, with a delivery ratio of 99.9% in all scenarios. Not even Epidemic Routing (blue) achieves this level of success for high load scenarios, because it produces the highest load and can easily overload the network. Furthermore, PRoPHET (orange) is ill-suited for this scenario and achieves only mediocre results (depending on the experiment, between 43% and 70%). This does not mean that PRoPHET is a bad routing algorithm, but if a scenario does not conform to its assumptions, it will fail to achieve its intended result.

Delivery Times. The next metric is *delivery time*, i.e., the time it takes for a bundle to reach its intended recipient. For broadcasts, we consider the delivery time to the first eligible recipient. The results are shown in Fig. 4; the x-axis

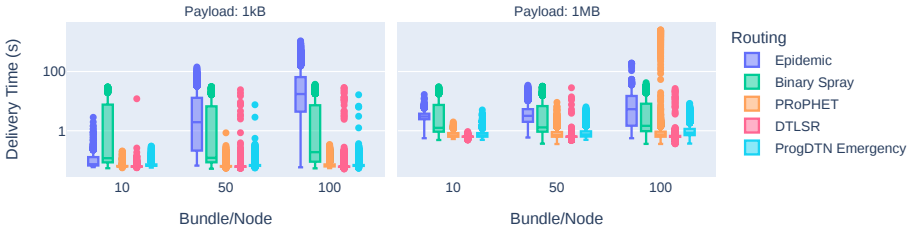

Fig. 4: Time to deliver a bundle to its destination for different parameters

shows bundles per node, the y-axis the delivery time on a logarithmic scale, the color denotes the routing algorithm, and each sub-figure shows the different payload sizes. ProgDTN Emergency shows results at least on-par with the other algorithms, with a median below 1 second, regardless of the scenario, and rare cases where the delivery time exceeds 15 seconds for high load scenarios. Epidemic Routing loses performance in higher-load scenarios due to excessive network load. In extreme cases, a bundle can take up to 15 minutes to reach its destination. However, the decrease in long-time outliers for the higher-payload scenario is because we only see initial, fast deliveries for this scenario, while once the system reaches congestion, bundles do not arrive at all and are thus absent from this graph. This observation is also consistent with the observation made for the delivery ratios. Under certain conditions, PRoPHET behaves quite erratic (one hour delivery time), because delivery probabilities are only updated for new connections, which leads to race conditions. This shows that PRoPHET it is not well suited for this scenario. However, all these variations of epidemic routing and PRoPHET are outliers, while the 75% quantile remains quite small, e.g., about 17 seconds for epidemic routing and 1 second for PRoPHET.

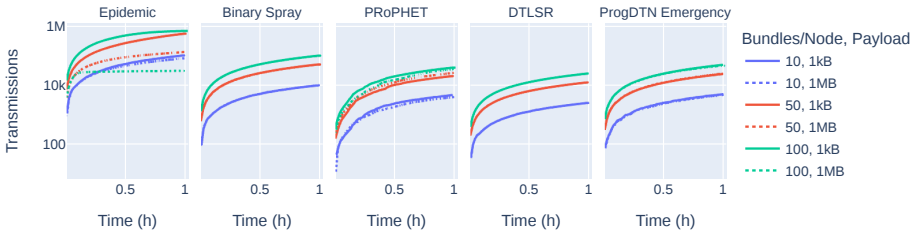

Fig. 5: Total number of bundle transmissions for different parameters

Network Load. Fig. 5 shows the number of bundle transmissions throughout the experiment, where the x-axis shows the time and the y-axis the transmissions on a logarithmic scale. The color denotes the bundles per node, the line style and payload size, and each sub-plot shows a routing algorithm. From Fig. 5 it becomes apparent that Epidemic Routing produces orders of magnitude more transmissions than all other algorithms, i.e., up to 695,000 bundles over one hour compared to about 50,000 bundles for ProgDTN Emergency and 25,000 bundles

for DTLSR for 100 bundles per node and a payload size of 1 kB. The cause is the inefficiency of Epidemic Routing; it replicates all bundles to all peers without any concern for whether a transmission increases the delivery probability. Binary Spray & Wait also produces a relatively high load level compared to algorithms other than Epidemic Routing, while the remaining algorithms have a somewhat similar load level. ProgDTN Emergency successfully avoids unnecessary transmissions and conserves network capacity. If this data is viewed in conjunction with Figs. 3 and 4, it is apparent that ProgDTN Emergency achieves high average delivery ratios of about 99.9% within 1 second, while other routing algorithms suffer in different quality metrics for various reasons.

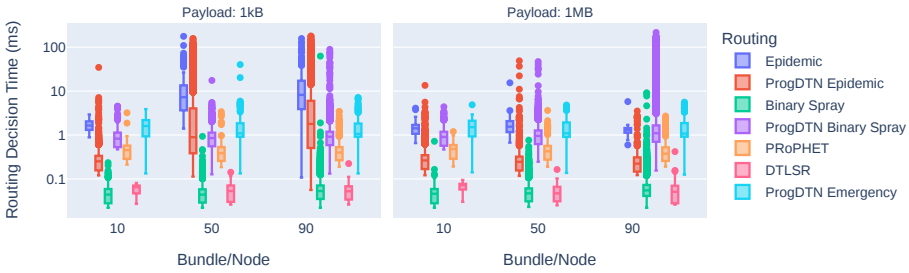

Fig. 6: Time to make a routing decision for different parameters

Routing Decisions. Fig. 6 shows the time it takes to perform a routing decision for each algorithm. The x-axis groups different bundles per node, and the y-axis shows the time in ms it takes the routing algorithm to finish on a logarithmic scale. Each color represents a routing algorithm, while the two subplots show the results for different payload sizes. The focus of this metric is the comparison between Epidemic Routing and ProgDTN Emergency and between Binary Spray & Wait and ProgDTN Binary Spray & Wait, since it shows the overhead introduced by the JavaScript VM. It is evident that the ProgDTN variants of the two algorithms take longer for their routing decisions compared to the non-ProgDTN variants. This is not surprising, since every time a ProgDTN-based algorithm has to make a routing decision, it needs to initialize and start a JavaScript VM and then execute the actual routing code, which is interpreted rather than run as native code. However, the mean and 75% quantile is still well below 50 ms even in those cases. Furthermore, ProgDTN Emergency performs reasonably well with a median of about 1.5 ms and a 75% quantile of about 3 ms, regardless of the experiment. To summarize, the JavaScript VM introduces overhead that is compensated by the fact that ProgDTN can be used to implement a scenario-specific algorithm, reducing the average time to deliver a bundle.

Bundle Overhead. Since ProgDTN makes use of context bundles to let nodes exchange context information (see Section 3.3), the amount of additional traffic generated by these bundles needs to be quantified, since it may contradict the qualitative metrics of preserving network bandwidth. Fig. 7 shows the overhead

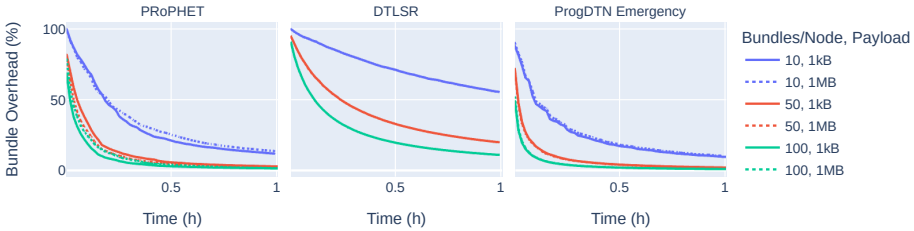

Fig. 7: Overhead in terms of percentage of bundles sent without a payload

of all algorithms that transmit metadata bundles in terms of the percentage of sent bundles, where the x-axis denotes the experimental runtime, the y-axis the percentage of context bundles (in case of DTLSR so-called meta bundles). The color shows different bundles per node, and the line style represents the two payload sizes. Finally, each sub-figure shows a different routing algorithm. Note that only the three shown routing algorithms produce an overhead. Since each node sends a context or meta-bundle upon startup for all algorithms, they all start at 100% overhead; this percentage does, however, decrease rapidly as payload bundles start being transmitted. For both PRoPHET and ProgDTN Emergency, we see an exponential decrease with the overhead percentage converging to zero, but remember that PRoPHET does not achieve a satisfactory delivery ratio in this scenario. DTLSR, on the other hand, regularly broadcasts peer information to the whole network, so we see a higher overhead throughout the experiment.

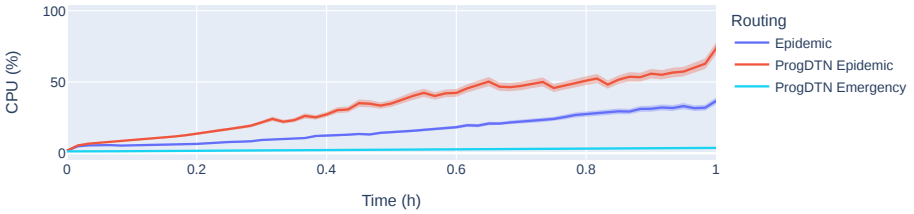

Fig. 8: CPU usage of three routing algorithms

CPU Usage. Fig. 8 shows the overhead in terms of CPU usage. The x-axis denotes the experimental time. The y-axis shows the average CPU usage in percent of all nodes. The colors represent different routing algorithms. For this evaluation, we only consider Epidemic Routing, ProgDTN Epidemic, and ProgDTN Emergency to quantify the overhead of the CPU and the potential savings using a custom routing algorithm. The blue curve, representing Epidemic Routing, gives a baseline for system load that correlates with network load shown in Fig. 5. Since the network saturation increases over the experimental run, the system load increases in step. The red curve shows the re-implementation of Epidemic Routing in ProgDTN. As expected, it introduces a higher CPU usage, which is due to the increased computational load introduced by the constant

re-initialization of the JavaScript VM. The custom routing algorithm, as represented by the turquoise line, by contrast, shows the smallest CPU load, even though it also makes use of the same system as ProgDTN Epidemic. The reason is that if there are fewer bundles to transmit, the JavaScript VM is invoked less often, reducing the overall system load due to fewer transmissions. Thus, by reducing the number of transmissions, we can conserve computing power and therefore also electrical power in cases where a device might be battery-powered.

To summarize, ProgDTN Emergency achieves excellent results and outperforms all other routing algorithms in all metrics. The results of PRoPHET show that a routing algorithm not designed for a particular scenario does not achieve any satisfactory results. Epidemic Routing achieves good results for smaller payload sizes and a small numbers of bundles per node, but its delivery ratios decrease below 50% for high network load, while ProgDTN Emergency still achieves 99.9%. DTLSR also achieves excellent delivery ratios and delivery times, but its overhead in terms of meta-bundles is significant compared to ProgDTN Emergency. Finally, Binary Spray & Wait performs considerably well, but requires more transmissions to achieve comparable results to ProgDTN Emergency.

6 Conclusion

We presented *ProgDTN*, a novel approach to support *Programmable Disruption-tolerant Networking* by allowing network operators to program a node's routing behavior based on context information, without requiring knowledge of the router's interior workings. Our experimental evaluation showed that a programmable DTN routing algorithm tailored to a specific scenario achieves excellent results in terms of up to 99.9% delivery ratio while reducing unnecessary transmissions by 92.9% compared to state-of-the-art DTN routing algorithms in an emergency response scenario. We achieved a low delivery time of bundles (1 – 15 seconds), and low overhead in terms of CPU utilization and routing decisions.

There are several areas of future work. First, implementing a system that allows updating or replacing routing algorithms at runtime would reduce unnecessary downtimes and further reduce development and deployment hurdles. Second, allowing a centralized entity to reconfigure an entire DTN deployment would make the administration and monitoring of DTN nodes more flexible. Finally, although a network emulation gives valuable insights, evaluating ProgDTN on real hardware in a real mobile scenario would further solidify and confirm the applicability and feasibility of ProgDTN.

Acknowledgements

This work is funded by the Hessian State Ministry for Higher Education, Research and the Arts (HMWK) (LOEWE emergenCITY, LOEWE Natur 4.0), and the German Research Foundation (DFG, Project 210487104 - Collaborative Research Center SFB 1053 MAKI).

References

1. Ahrenholz, J.: Comparison of CORE network emulation platforms. In: MILCOM 2010 Military Communications Conference. pp. 166–171 (Oct 2010). https://doi.org/10.1109/MILCOM.2010.5680218
2. Baek, K.M., Seo, D.Y., Chung, Y.W.: An Improved Opportunistic Routing Protocol Based on Context Information of Mobile Nodes. Applied Sciences **8**(8) (2018). https://doi.org/10.3390/app8081344
3. Bansal, A., Gupta, A., Sharma, D.K., Gambhir, V.: IICAR - Inheritance Inspired Context-aware Routing Protocol for Opportunistic Networks. Journal of Ambient Intelligence and Humanized Computing **10**(6), 2235–2253 (Jun 2019). https://doi.org/10.1007/s12652-018-0815-2
4. Biswas, P.K., Mackey, S.J., Cansever, D.H., Patel, M.P., Panettieri, F.B.: Context-Aware Smallworld Routing for Wireless Ad-Hoc Networks. IEEE Transactions on Communications **66**(9), 3943–3958 (Sep 2018). https://doi.org/10.1109/TCOMM.2018.2811486
5. Boldrini, C., Conti, M., Jacopini, J., Passarella, A.: HiBOp: a History Based Routing Protocol for Opportunistic Networks. In: 2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks. pp. 1–12 (June 2007). https://doi.org/10.1109/WOWMOM.2007.4351716
6. Burleigh, S., Fall, K., Birrane, E.: Bundle Protocol Version 7. Internet draft, RFC Editor (March 2020), https://tools.ietf.org/html/draft-ietf-dtn-bpbis-24
7. Cabaniss, R., Madria, S., Rush, G., Trotta, A., Vulli, S.S.: Dynamic Social Grouping based Routing in a Mobile Ad-Hoc Network. In: 2010 International Conference on High Performance Computing. pp. 1–8 (Dec 2010). https://doi.org/10.1109/HIPC.2010.5713165
8. Demmer, M., Fall, K.: DTLSR: Delay Tolerant Routing for Developing Regions. In: Proceedings of the 2007 Workshop on Networked Systems for Developing Regions. pp. 5:1–5:6. NSDR '07, ACM, New York, NY, USA (2007). https://doi.org/10.1145/1326571.1326579
9. Er-rouidi, M., Moudni, H., Faouzi, H., Mouncif, H., Merbouha, A.: A Fuzzy-Based Routing Strategy to Improve Route Stability in MANET Based on AODV. In: Networked Systems. pp. 40–48. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-59647-1_4
10. Froemmgen, A., Stohr, D., Koldehofe, B., Rizk, A.: Don't Repeat Yourself: Seamless Execution and Analysis of Extensive Network Experiments. In: Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT'18) (2018). https://doi.org/10.1145/3281411.3281420
11. Johari, R., Gupta, N., Aneja, S.: CACBR: Context Aware Community Based Routing for Intermittently Connected Network. In: Proceedings of the 10th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, &; Ubiquitous Networks. pp. 137–140. PE-WASUN '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2507248.2507272
12. Lindgren, A., Doria, A., Schelén, O.: Probabilistic Routing in Intermittently Connected Networks. In: Service Assurance with Partial and Intermittent Resources. pp. 239–254. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1145/961268.961272
13. Mascolo, C., Musolesi, M.: SCAR: Context-aware Adaptive Routing in Delay Tolerant Mobile Sensor Networks. In: Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing. pp. 533–538. IWCMC '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1143549.1143656

14. Musolesi, M., Hailes, S., Mascolo, C.: Adaptive Routing for Intermittently Connected Mobile Ad Hoc Networks. In: Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks. pp. 183–189 (June 2005). https://doi.org/10.1109/WOWMOM.2005.17
15. Musolesi, M., Mascolo, C.: CAR: Context-Aware Adaptive Routing for Delay-Tolerant Mobile Networks. IEEE Transactions on Mobile Computing **8**(2), 246–260 (Feb 2009). https://doi.org/10.1109/TMC.2008.107
16. Penning, A., Baumgärtner, L., Höchst, J., Sterz, A., Mezini, M., Freisleben, B.: DTN7: An Open-Source Disruption-tolerant Networking Implementation of Bundle Protocol 7. In: International Conference on Ad-Hoc Networks and Wireless (AdHoc-Now 2019). pp. 196–209. Springer, Luxembourg, Luxembourg (2019). https://doi.org/10.1007/978-3-030-31831-4_14
17. Rosas, E., Garay, F., Hidalgo, N.: Context-aware Self-adaptive Routing for Delay-tolerant Networks in Disaster Scenarios. Ad Hoc Networks **102**, 102095 (2020). https://doi.org/10.1016/j.adhoc.2020.102095
18. Spyropoulos, T., Psounis, K., Raghavendra, C.S.: Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In: Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking. pp. 252–259. WDTN '05, ACM, New York, NY, USA (2005). https://doi.org/10.1145/1080139.1080143
19. Ullah, S., Qayyum, A.: Socially-aware Adaptive Delay-tolerant Network (DTN) Routing Protocol. PLOS One **17**(1), 1–15 (01 2022). https://doi.org/10.1371/journal.pone.0262565
20. Vahdat, A., Becker, D., et al.: Epidemic Routing for Partially connected Ad Hoc Networks. Tech. Rep. CS-200006, Duke University (2000)