DTN Hybrid Networks for Vehicular Communications

Justin P. Rohrer and Geoffrey G. Xie
Department of Computer Science
Graduate School of Operational & Information Sciences
Naval Postgraduate School
Monterey, CA 93943–5285
{jprohrer|xie}@nps.edu

***Abstract*—We present an architecture for combining two established network paradigms, IP and Disruption-Tolerant Networking (DTN), into a unified packet gateway design that leverages the advantages of both. Vehicular networking (VNET) scenarios often involve brittle links between communicating nodes due to their mobility. DTN solutions, by using a dynamic hop-by-hop delivery model instead of the end-to-end IP model, are able to sustain a large class of applications despite intermittent links. As a defining characteristic, our design is *application-transparent* in that it requires no changes to host applications (or the underlying host protocol stacks) in order for them to use DTN transport when IP is not feasible. In addition, we build into the architecture an explicit disruption notification service for keeping users informed as well preventing application time-outs during an IP outage. Finally, given the wide range of behaviors exhibited by applications that can benefit from DTN, our design supports the notion of an *application lattice* to allow operators to customize, on a per application/group/protocol basis, how the switch between IP and DTN and the disruption notification are performed. A preliminary evaluation based on a C++ proof-of-concept implementation has illustrated several potential benefits of the proposed architecture for VNET applications.**

I. Introduction

As ubiquitous access to information is ever critical in the information age, network deployments are rapidly expanding into dynamic ad hoc environments that are not well supported by the standard IP functionality. In particular, a variety of applications have been conceptualized over vehicular networks (VNETs) [1], [2], where both the connectivity between vehicles and the access links to the IP backbone infrastructure can be brittle. While IP would stop working upon the absence of an end-to-end communication path, alternative solutions such as disruption/delay tolerant networking (DTN) [2] are specifically designed to work in VNET like challenging environments with a hop by hop dynamic delivery strategy.

However, the entrenched application ecosystem is exclusively IP-based. As such, a fundamental question is how to integrate non-IP and IP networks into one infrastructure in an *application transparent* fashion, which requires no changes to applications and furthermore, minimizes the potential disruptions to applications.

The predominant approach to integrating non-IP networks follows a vertical overlay model. In the case of DTN, it is either IP-over-DTN or DTN-over-IP. This layered approach is simple to design and implement, by requiring no additional data translation module. However, it forces least-common denominator semantics for transport of data across network boundaries, and as such, may greatly hamper the working of applications originally designed for an IP network. Specifically, there is a prevailing perception that the DTN technology is not plug-and-play and existing applications must be retrofitted to use DTN. Consequently, while DTN has been repeatedly demonstrated to be beneficial in many scenarios involving challenged networks, its deployment is still very limited even after more than a decade of refinement.

To further motivate the need of an application transparent approach, we identify three common vehicular-network edge-scenarios that would benefit from DTN technology. All three scenarios require seamless, dynamic integration of IP or DTN transport.

- **Episodic connectivity:** IP ceases functioning entirely when it cannot find an end-to-end path, consequently causing applications (e.g., Web and map download) to time out. In contrast, DTN buffers data (called bundles) at an intermediate node when a next hop is temporally unavailable, and as such is able to sustain applications as long as a sequence of one-hop forwarding can reach the destination eventually.
- **Degraded link-quality:** TCP responds adversely to dropped or corrupted packets, bringing the performance of TCP-based applications to a standstill in severe cases. By using DTN's hop-by-hop error-correction capability this loss in application performance can be significantly reduced.
- **Policy-driven prioritization:** As vehicular networks move toward providing differentiated quality-of-service to prioritize certain traffic classes, it is possible for a sustained burst of high-priority traffic to starve out a lower priority flow, resulting in essentially the same effect for that flow as an episodically-connected link. By the time link capacity is again available the application will

DOI 10.1109/ICCVE.2013.25

have timed out and will not be able to make use of that available capacity. With DTN in the network, lower priority traffic can be bundled and delivered as soon as link capacity is again available, resulting in higher overall application performance.

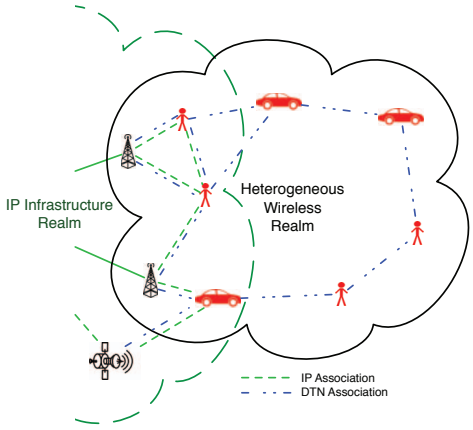


Fig. 1: IP-*cum*-DTN scenario

In this paper, we present a new approach to integrating IP and DTN, which we term "IP-*cum*-DTN" (where *cum* is a Latin term for *with and along side*). IP-*cum*-DTN combines the two paradigms in a unified architecture that leverages the advantages of both. Specifically, we introduce an additional architectural component to splice IP and DTN functionalities into a single logical network layer. Our architecture can be implemented as a gateway that intelligently selects between native IP and DTN network options, based on application characteristics. Essentially, the gateway provides a service to applications, which we term the *Application-aware Dynamic Network Selection (ADNS)* service. ADNS intercepts live packets in the IP buffer immediately upon network disruption events and selectively bundles some of these packets for transport with DTN according to preconfigured application-specific policy. The delay tolerance of applications is profiled *a priori*. A formal *policy lattice* model is introduced to capture the result and precisely define, for each application, the required DTN forwarding mechanism, *if any*, the gateway should use upon a network disruption event.

Our design targets a broad range of networking scenarios where link connectivity is increasingly heterogenous at the edge and realms of disparate technologies [3] coexist to address scenario-specific requirements. We observe that while IP realms are circumscribed by a hard edge where infrastructure coverage ends, DTN realms have no such defined border, extending as-needed even to space [4]. Therefore, we seek to blur this distinction between IP and DTN networks, by enabling connectivity for applications that were originally designed for IP beyond the confine of IP realms, as illustrated in Figure 1.

We have conducted a preliminary evaluation of our design on a small scale VNET that is being fielded by the United States Marine Corps. The results confirm that IP-*cum*-DTN is able to provide DTN support in an application-transparent manner. Wireless nodes of this network must venture into geographic areas that do not have infrastructure coverage, depending on satellite and terrestrial wireless links which may be disrupted due to any number of adverse conditions, and where the ability to access information can be crucial to life-or-death situations. While satellite links were the expected solution to most of these cases, the astronomical cost-to-bandwidth ratio has limited deployment and capacity is exceedingly limited. This is leading to greater use of short-range terrestrial wireless solutions which are frequently subject to either complete disconnection, or worse have faint and sporadic connectivity that is sufficient for applications to initiate connections, but inadequate for any meaningful communications. It is precisely these conditions that DTN networks are intended to mitigate, with the barrier of adoption consisting primarily in their inability to support legacy IP applications.

II. Background and Related Work

Bundling Protocols. The IETF delay-tolerant networking research group (DTNRG) has standardized two main DTN protocols, the Bundle Protocol [5] and the Licklider Transmission Protocol (LTP) [6]. The Bundle Protocol supports an overlay store-and-forward network that sends packages of application data – called bundles – over a wide range of underlying network types using a sequence of gateways that serve as nodes in the overlay network. This represents the mainstream approach within the DTNRG group. Example implementations include the SPINDLE 3 (BBN) [7], which we have been experimenting with on our testbed and several others recently compared by Pöttner et. al. in [8]. LTP is a point-to-point protocol that deals with individual long delay links by freezing timers that would otherwise expire before an acknowledgement was received. LTP does not handle congestion or routing issues [9].

Non-IP DTN Protocols. An alternative to using native IP or application-layer overlays, is to translate data into a custom protocol stack. A recent approach using this method is the ANTP suite [10], [11], which is composed of the AeroTP transport layer [12], [13], the AeroNP network layer [14], and the AeroRP routing layer [15]. While we do not expect to make use of these non-IP protocols, this work is relevant to the project because of the many parallels between the telemetry scenario and our use cases, as well as the fact that a method of realm-splicing between the ANTP suite and traditional IP protocols was developed in the form of the AeroGW gateway [16].

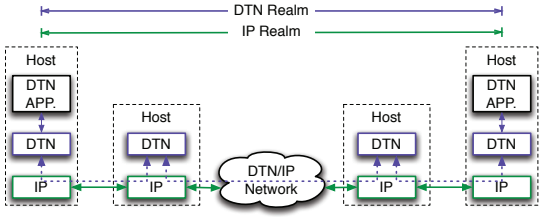
Connection Splicing. End-to-end (transport) connections must be spliced at realm boundaries. In unreliable networks, TCP performs poorly due to the high number of end-to-end retransmissions incurred and the congestion avoidance mechanism that is triggered by packet loss. Several TCP splicing methods have been proposed in the literature and similar concepts can be leveraged in an IP-*cum*-DTNgateway. **(i)** Split-TCP divides long paths into several shorter ones, inserting proxies to interface between the segments. The proxies buffer, acknowledge, and retransmit packets and are able to improve performance by

breaking up the end-to-end semantics of TCP [17]. **(ii)** Mobile TCP (M-TCP) makes use of a gateway that connects multiple cells to the fixed network to split the TCP connection. The standard TCP is used on the wired side while M-TCP is used between the mobile host and the gateway. If disconnected, the gateway advertises a receiver window of 0 to the fixed peer, putting it into persist mode. When reconnected, the gateway advertises the normal size window to the sender, allowing the connection to resume with no back-off [18]. **(iii)** Freeze-TCP is designed to improve TCP performance between mobile devices without splitting the connection or requiring changes to the TCP code on the fixed node. When the mobile device is the receiver, it uses signal-strength information from the device's radio to predict a disconnection or handoff and advertises a zero window size just before this happens. This forces the TCP sender in the zero-window-probing mode. To resume, the mobile receiver sends 3 ACKs for the last packet received to initiate fast-retransmit [19].

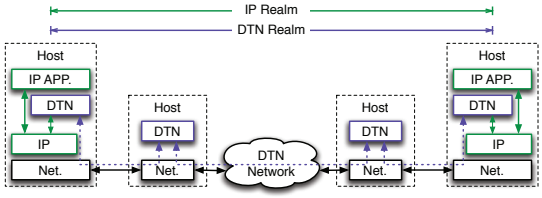
III. IP-*cum*-DTN Architecture

In this section we describe the design of the IP-*cum*-DTN architecture. The architecture supports the creation of novel network-level functionalities to mitigate unnecessary negative impacts on legacy applications while enabling the performance advantages of DTN networks. The details of one such functionality will be presented in Sections IV.

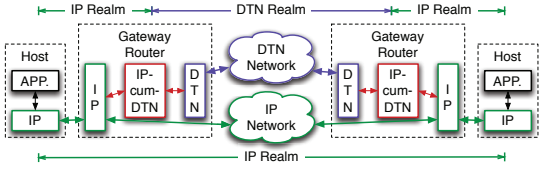
To motivate our design, we first contrast a layered model of our architecture to those of alternative architectures proposed in the literature. After presenting the design details of the IP-*cum*-DTN architecture, we conclude the section by describing some of our preliminary prototyping work.



(a) DTN-over-IP layer architecture



(b) IP-over-DTN layer architecture



(c) IP-*cum*-DTN layer architecture

Fig. 2: Alternative DTN/IP architectures

A. Alternative Architectures

Along with the maturity of both IP and DTN networks comes existing architectural designs for integrating the two, which fall into either an IP-over-everything (Figure 2b) or a DTN-over-everything (Figure 2a) model. Neither of these has been successful in enabling widespread adoption of DTN technology. We assert as an alternative, concurrent IP and DTN realms (Figure 2c) with intelligent selection at adjacent borders.

In the DTN-over-IP case shown in Figure 2a (or DTN-over another protocol), the full benefits of the DTN protocol are available, however since DTN is an application-layer overlay it is only compatible with applications written specifically to work with it. Other disadvantages to this approach include the overhead incurred by the bundling protocol end-to-end, and that the DTN routing does not interact with IP routing so the overlay network formed is likely to be far from optimal in the sense of shortest-path routing.

The IP-over-DTN case shown in Figure 2b builds on the DTN-over-IP scenario by encapsulating IP packets in DTN bundles, enabling conventional IP-based applications to benefit from DTN. Apart from the obvious layer violations involved in this approach it incurs significant overhead of the 52-byte DTN-header and in many cases a second 20-byte IP-header since DTN is most commonly run on top of IP networks. This is in addition to the drawbacks, as explained above, of incurring the overhead end-to-end when only a small subset of hops may experience disruptions, and of sub-optimal routing topologies. It also requires a DTN agent to run on every endpoint, even if some only send and receive IP traffic.

In contrast, the IP-*cum*-DTN architecture, as shown in Figure 2c, does not overlay protocols; instead, it relies on a third module in the gateway to splice the two realms into a single end-to-end layer, with context-appropriate capabilities. IP is used at the edge to support IP-based applications, and if coherent end-to-end paths through the network exist the gateway will keep the packets in the IP-realm end-to-end thus avoiding the overhead of the DTN bundling protocol and utilizing the IP-optimized established network infrastructure. However if the path is disrupted, the gateway will dynamically select the DTN-realm, translating the traffic as necessary. Since this occurs within the network at the boundary of the affected realm, the additional overhead of the bundling protocol is only incurred locally instead of penalizing every link on the end-to-end path, while still providing the benefits of DTN where needed. It also synchronizes the DTN and IP routing protocols, instead of ignoring IP routing as occurs in the DTN overlay scenarios. Additionally, because this gateway is adjacent to the disrupted region, it is in an ideal position to host a Disruption Notification Service.

B. System Architecture

Due to the maturity of both the DTN and IP microcosms our goal is to leave both as intact as possible, and introduce a lightweight translation-layer and decision plane as a software module that is external to both existing components, but

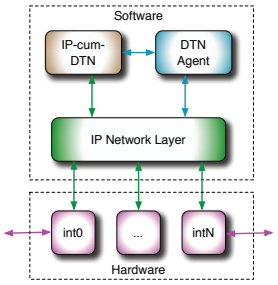


Fig. 3: IP-*cum*-DTN gateway architectural components

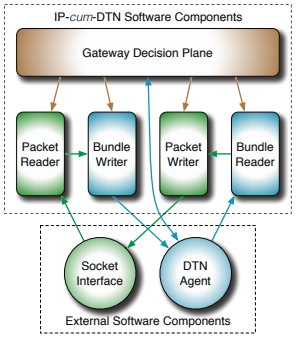


Fig. 4: IP-*cum*-DTN software components

triggered when IP packets arrive for which the IP next-hop is unavailable. The decision plane may incorporate a two-phase approach in which the triggering mechanism and a coarse-grained binary decision (to use DTN or not) is performed in the kernel space. The remaining packets are dispatched to different handling modules (likely in the user space) according to additional preconfigured application-specific policy; the details of defining this policy will be discussed in Section IV-A. Figure 3 shows the relationships between the IP-*cum*-DTN software module, the DTN Agent (also running in user-space), and the IP layer (typically embedded in the system kernel). It should be noted that while this figure is representative, there are also cases where the DTN agent uses non-IP network protocols to send bundles via the physical interfaces (int0...intN in the figure).

This module will also be responsible for splicing the DTN and IP control-planes together. We will provide both a static and a dynamic mechanism for this. The user will be able to statically configure IP networks to be associated with each DTN node, and the splicing agent will be able to learn IP networks to which it is connected from dynamic routing protocols. In the latter case these IP network prefixes will be periodically disseminated to neighboring DTN nodes so that a DTN-EID to IP-prefix mapping may be maintained. These mechanisms will allow each bundle created from translated IP packets to be addressed for delivery to the correct DTN host.

A potential IP-*cum*-DTN packet-pipeline consists of the following steps (corresponding to Fig. 4): IP packet received by kernel; packet passed to *packet reader*; *packet reader* queries *decision plane*; *decision plane* determines suitability for DTN forwarding and initiates any needed response via *packet writer*; if approved *packet reader* queues the packet for *bundle writer*; *bundle writer* queries the decision plane for next-hop DTN EID (Endpoint Identifier) and aggregates with packets to same EID; *bundle writer* generates bundle header and passes to DTN agent. The reverse process includes fewer steps because we do not egress filter and IP semantics are preserved. Again referring to Figure 4 the potential steps include: DTN bundle received by *DTN Agent*; bundle requested by *bundle reader*; bundle payload decomposed into IP packets, which are passed to the *packet writer*; packets passed to the kernel for conventional IP routing and forwarding.

IV. Application-Aware Dynamic Network Selection

Applications exhibit a wide spectrum of tolerance to network delays. The IP-*cum*-DTN gateway should leverage this information to customize their DTN forwarding behavior. For example, a real-time interactive application such as VoIP does not benefit from DTN and its packets should be *dropped* (or *re-routed* to a new IP destination) when an IP path for the original destination is not available. Additionally, some delay-tolerant applications (e.g., chat) are TCP based. Their performance may suffer because of TCP connection timeout events. The gateway should prevent such events by injecting special signaling packets into the TCP connections.

In order to manage complexity, we use a systematic framework to enable customization of packet handling according to pre-configured application-specific policy, which can even be customized on a per gateway basis. Instead of a tree structure commonly used for firewall design, we have chosen a lattice structure for policy representation because the latter facilitates greater reuse of nodes at each layer. The application profile lattice (Figure 5) is created *a priori* by profiling the set of applications expected to be deployed in the IP-*cum*-DTN network. Each lattice node indicates a specific set of actions to be taken, and each link indicates a conjoining operation between the actions at each level of the lattice. The path through the lattice is determined by fields in the packet header, predominantly the IP Protocol ID field and the TCP/UDP port number, however application-specific fields may be used as well. The lattice must meet meet some basic criteria:

- **Completeness:** It must be complete, i.e. there must be a defined path through the lattice for all possible input packets.
- **Exactness:** It cannot be ambiguous, i.e. the entry criteria for all nodes at a given level must be orthogonal.
- **Conflict-free:** It also must have a defined conflict-resolution policy, e.g. lower-level (more refined) actions supersede higher-level (more coarse-grained) actions.

As part of this work we will generate lattices for a general set of applications commonly used in mobile environments.

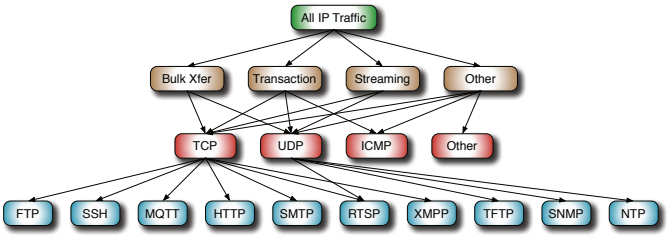


Fig. 5: Application-profile lattice

A. Application Profiling

As shown in Figure 5 we categorize applications by profiling their network activity according the standard categories: bulk transfer, transactional (interactive), and streaming. We then subdivide these categories based on the transport layer used (e.g. TCP, UDP, SCTP), and finally by individual application,

thus forming a lattice where the root is IP and individual applications are the leaves. Our hope is to constrain the complexity of the solution by reusing as much packet-handling code as possible. For example if a method works on multiple TCP-based bulk-transfer applications it can be implemented at the protocol layer of the lattice and ideally only a subset of the applications will need any unique handling beyond that layer.

B. Modular Handling Methods

The software framework will be modular, with each node in the lattice indicating actions, which are handled by a plugin for that category, protocol, application, etc. Additionally there will be a catch-all module for handling packets not associated with a profiled application. This is possible by supporting mechanisms at varying granularity within the application lattice. Node actions can range from the very simple to sophisticated, perhapse applying only to a single application-layer message type, or an entire transport-layer protocol. An example simple action is *drop*, most likely called for in cases where delay makes delivery of a particular type of traffic no longer worthwhile. Nearly as simple is the *buffer* action, which invokes the IP-to-DTN translation component of the architecture and bundles the traffic for DTN handling. More complex actions are called for when a particular protocol or application needs special treatment to avoid timing out too quickly. Part of this handling will involve initiating ENDN messages (described further in Section IV-D), however it will take time for the benefits of this explicit signaling to be incorporated into applications. On the other hand significant work has been accomplished in the area of making various protocols tolerant of delay, and we will build on this body of existing research to create advanced actions. For example, the Freeze-TCP [19] method allows us to "pause" a TCP flow and can be incorporated into the lattice either as a protocol-layer action to be applied to all TCP traffic, or as part of an application-specific response, applying only to a subset of TCP traffic.

As IP packets arrive at the gateway they will be routed to the appropriate module based on standard port numbers and application headers, as well as deployment-specific port numbers specified in the configuration file. The configuration file(s) will be structured, with sections for each module, and including plugin-specific configuration files which may be added ad-hoc.

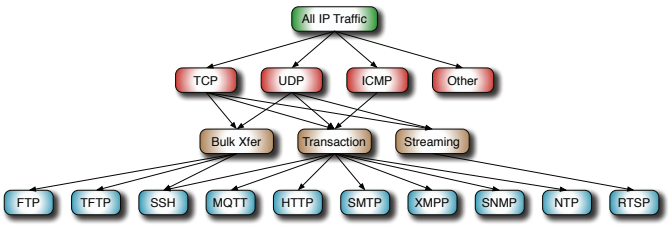


Fig. 6: Alternate application-profile lattice

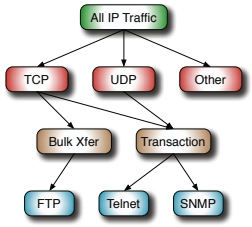


Fig. 7: Simple example application-profile lattice

C. Examples

Many alternative lattices can be envisioned and we will look at some of these as well. For example greater efficiencies might be obtained by switching the middle layers as shown in Figure 6. A controlled-access network (e.g. sensor/telemetry, military, or deep space) might have a very narrow lattice as shown in Figure 7, where the only applications allowed on the network are FTP, Telnet, and SNMP. All IP traffic is categorized into TCP, UDP, or Other. When a packet that is part of an FTP flow arrives, it will match the *TCP* node where the action to occur is setting the *custody-transfer* DTN bundle flag. The FTP packet is then routed to the *Bulk Transfer* node where the *bulk* priority flag is set for the DTN bundle. Lastly the FTP packet is routed to the *FTP* node, which first checks the available DTN buffer size. If the size is below a threshold an FTP server emulator is initiated that acknowledges the packets so that the file will continue to be transferred to the DTN buffer. If the buffer is getting full, the the *FTP* node initiates the Freeze-TCP algorithm [19] so that the transfer will not time out and can be resumed when buffer space is available or an end-to-end path is re-established.

When a Telnet packet arrives it will again be routed to the *TCP* node where the same action applies, however it will then be routed to the *Transaction* node where the *expedited* priority flag is set for the DTN bundle. Finally it is routed to the *Telnet* where the ENDN notification service (Section IV-D) is initiated using the appropriate delay code. An SNMP packet is routed to the *UDP* node first, where no action is indicated. It is then routed to the *Transaction* node with the same action as before. Lastly it reaches the *SNMP* leaf where no action is taken. In practice this leaf would be eliminated since no application-specific handling is required.

Any packets belonging to protocols other then TCP or UDP are routed to the *Other* node, where they are dropped, and an ENDN response sent with the Explicit Loss Notification message code as discussed in Section IV-D). Packets that are TCP or UDP, but do not belong to one of the three permitted applications are handled likewise.

D. Network Disruption Notification

Clearly there are limits to the length of delays that can be handled transparently in the network, and that is dependent on the application category being handled. Eventually application-level timeout will occur, or the user will get tired of waiting without seeing any apparent activity. Therefore we submit that an explicit notification mechanism is required

for delay-tolerance to be seamless in heterogeneous-delay environments.

There are several viable approaches to implement such a service, possibly leveraging ECN bits (for TCP traffic), the path-MTU discovery mechanism, a new HTTP code, or an ICMP/ICMPv6 extension. We leave further exploration of this topic to future work, but our intuition is that this service should be able to deliver a rich set of information back to the source application or user about the type and expected severity of the loss or delay.

V. Preliminary Evaluation

We have built several Linux-based IP-*cum*-DTN gateway prototypes, and used them to establish a testbed for DTN experimentation, designed to be suitable both for lab experimentation as well as vehicular deployment for field experimentation.

The DTN gateway prototype hardware is based on the AMD Brazos platform, chosen for its high I/O-bandwidth capability, low cooling requirements, and high performance vs. cost efficiency. System storage and DTN buffering are provided using high-speed synchronous flash, accessed via a 6.0 Gbit/s serial ATA interconnect. In addition to the onboard gigabit ethernet interface, we provide four additional routable gigabit interfaces and eight gigabytes of DRAM. The entire system is enclosed in a steel chassis 8.7" wide, 12.9" deep, and 3.8" tall, as shown in Figure 8. Power consumption is approximately 35 W under typical load, which is low enough that active cooling fans are not required under most circumstances. The power supply requires a 12v power source, which may be either a laptop-type power brick, or a standard automotive auxiliary power source. The resulting device is relatively small, consumes little power, and requires little cooling, making it suitable for mobile platforms.

Fig. 8: NPS DTN gateway hardware

We base our software development on the Linux software routers derived from the Debian distribution [20]. For IP routing we utilize the Quagga [21] routing implementation to support major dynamic routing standards including OSPFv2 [22], OSPFv3 [23], RIPv2 [24], RIPng [25], and BGPv4 [26]. Of these OSPFv3, RIPng, and BGPv4 include support for IPv6. This platform provides high-speed, stable IP packet forwarding through the network as long as coherent end-to-end paths exist. We are evaluating the ION DTN agent developed by JPL, SPINDLE3 (BBN), and the DTN2 (Community) DTN agent for stability and suitability for our purposes. The bundling agents provide a plug-in interface for routing modules, allowing us to experiment with state-of-the-art DTN routing protocols.

We have been successful in implementing the software architecture shown in Figure 4 on our testbed implementation using a socket instance of the TUN interface as the triggering mechanism. This is a virtualized interface provided by the Linux operating system, instances of which can be created on demand. Using this method we have been able to intercept packets from various applications including Ping and Chat, encapsulate them in DTN bundles, forward them over the DTN network, un-bundle and deliver them to their IP destination. Using the ioctl system calls we are able to programmatically manipulate the IP routing table to intelligently determine what packets are intercepted for bundling.

In our preliminary testing we found the system to be capable of simultaneously routing two flows of 800 Mb/s each, effectively saturating the unidirectional capacity of 4 of the 1 Gb/s interfaces. This represents a routed traffic load of over 120,000 pkts/s. We used OSPF for route discovery and enabled the DTN bundling agent during these tests. While under this traffic load we observed that the average system load remained below 2%, and system memory usage remained below 150 MBytes out of the available 8192 MBytes. We will be continuing to increase the number of clients and DTN nodes in our testing environment, resulting in both additional routes and traffic flows, but these preliminary performance results indicate an ability for networks of our DTN gateways to scale well.

Recently the NPS DTN gateways were tested in the US Marine Core Network On The Move (NOTM) vehicular network environment and shown to provide reliable delivery under disrupted connectivity conditions. NOTM is a multihop mobile wireless environment with multiple short and mid-range radio technologies back-hauled by satellite links to only a few of the nodes. In this environment the DTN gateways were able to handle TCP and UDP traffic in a generic fashion, as well as provide enhanced support for specific applications, including the elimination of error-massages due to timeouts and user-visible notification of delayed message delivery. This was all handled in the gateway routers, without any change to the application or end host configuration.

One example of an application demonstrated on NOTM and benefitting from IP-*cum*-DTN is SIP (Session Initiation Protocol) Chat. Typically implementations of this protocol use UDP to transport bot data and control messages, and use short-duration timers to trigger retransmissions when data is unacknowledged. The typical exchange consists of a single UDP packet containing a message request header followed by the contents of the chat message. The response is a "202 Accepted" message within a few seconds. This reply does *not* imply receipt by the end user, only by an intermediary (often the receiver-side chat application) that will deliver the message to the end user. When the underlying network is disrupted the response does not come back and after a few retries the user is notified of the failure to deliver the message. When used in conjunction with the DTN gateway, the gateway generates the

"202 Accepted" message on seeing a chat message request and sends an additional chat message request back to the sender with a notification that their chat message was being delayed, but had not been lost. The 202 code prevents the multiple retransmission attempts by the sender's client, as well as the subsequent error message. The gateway then forwards the message when connectivity is reestablished, requiring no user intervention or application reconfiguration.

The benefits of incorporating IP-*cum*-DTN into NOTM include reduced overhead due to TCP end-to-end retransmissions, reduced loss of UDP traffic, and the prevention of timeouts by providing expected responses for buffered traffic.

VI. Future Work

We intend to further modularize our software architecture to reflect the lattice construct shown in Figure 5. In our current prototype each application is supported by a unique plugin. Our goal for the future is to create a fine-grained set of composable plugins so that a new application can be supported by writing a structured configuration file that identifies a set of these plugins and arranges them in a pipeline configuration. We are also interested in leveraging geolocation information for cross-layer usage. For example dynamically subscribing vehicular nodes to location-specific DTN multicast groups, to enable caching of application-specific data. Map, weather, and public safety alert data come to mind Additionally we intend to prototype a version of the ENDN service discussed in this work.

Acknowledgements

The authors would like to thank the US Marine Corps for their support. This work was funded in part by the Marine Corps Systems Command, Program Manager (PM) MAGTF Command, Control, and Communications (MC3).

References

[1] Y. Toor, P. Muhlethaler, A. Laouiti, A. Fortelle, and E. Mines, "Vehicle ad hoc networks: Applications and related technical issues," *IEEE Comm. Surveys and Tutorials*, vol. 10, no. 3, pp. 74–88, 2008.

[2] P. Pereira, A. Casaca, J. Rodrigues, V. Soares, J. Triay, and C. Cervello-Pastor, "From delay-tolerant networks to vehicular delay-tolerant networks," *IEEE Comm. Surveys and Tutorials*, vol. 14, no. 4, pp. 1166–1180, 2011.

[3] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks: Special Issue on Resilient and Survivable Networks (COMNET)*, vol. 54, pp. 1245–1265, June 2010.

[4] D. Brown, K. Trinidad, and R. Borja, "NASA successfully tests first deep space internet." http://www.nasa.gov/home/hqnews/2008/nov/HQ_08-298_Deep_space_internet.html.

[5] K. Scott and S. Burleigh, "Bundle Protocol Specification." RFC 5050 (Experimental), Nov. 2007.

[6] M. Ramadas, S. Burleigh, and S. Farrell, "Licklider transmission protocol–specification," Internet draft, Experimental draft-irtf-dtnrg-ltp, IETF, January 2008.

[7] R. Krishnan, P. Basu, J. M. Mikkelson, C. Small, R. Ramanathan, D. W. Brown, J. R. Burgess, A. L. Caro, M. Condell, N. C. Goffee, R. R. Hain, R. E. Hansen, C. E. Jones, V. Kawadia, D. P. Mankins, B. I. Schwartz, W. T. Strayer, J. W. Ward, D. P. Wiggins, and S. H. Polit, "The SPINDLE disruption-tolerant networking system," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, (Orlando, FL, USA), pp. 1–7, October 29–31 2007.

[8] W.-B. Pöttner, J. Morgenroth, S. Schildt, and L. Wolf, "Performance comparison of DTN bundle protocol implementations," in *Proceedings of the 6th Proceedings of the 6th ACM workshop on Challenged networks (CHANTS)*, pp. 61–64, September 23 2011.

[9] S. Farrell, V. Cahill, D. Geraghty, I. Humphreys, and P. McDonald, "When TCP breaks: Delay- and disruption- tolerant networking," *IEEE Internet Computing*, vol. 10, no. 4, pp. 72–78, July-Aug. 2006.

[10] J. P. Rohrer, A. Jabbar, E. Perrins, and J. P. G. Sterbenz, "Cross-layer architectural framework for highly-mobile multihop airborne telemetry networks," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, (San Diego, CA, USA), pp. 1–9, November 2008.

[11] J. P. Rohrer, A. Jabbar, E. K. Çetinkaya, E. Perrins, and J. P. Sterbenz, "Highly-dynamic cross-layered aeronautical network architecture," *IEEE Transactions on Aerospace and Electronic Systems (TAES)*, vol. 47, pp. 2742–2765, October 2011.

[12] J. P. Rohrer, E. Perrins, and J. P. G. Sterbenz, "End-to-end disruption-tolerant transport protocol issues and design for airborne telemetry networks," in *Proceedings of the International Telemetering Conference*, (San Diego, CA), October 27–30 2008.

[13] J. P. Rohrer and J. P. G. Sterbenz, "Performance and disruption tolerance of transport protocols for airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (Las Vegas, NV), October 2009.

[14] A. Jabbar, E. Perrins, and J. P. G. Sterbenz, "A cross-layered protocol architecture for highly-dynamic multihop airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 27–30 2008.

[15] A. Jabbar and J. P. G. Sterbenz, "AeroRP: A geolocation assisted aeronautical routing protocol for highly dynamic telemetry environments," in *Proceedings of the International Telemetering Conference (ITC)*, (Las Vegas, NV), October 2009.

[16] E. K. Çetinkaya and J. P. G. Sterbenz, "Aeronautical gateways: Supporting TCP/IP-based devices and applications over modern telemetry networks," in *Proceedings of the International Telemetering Conference*, (Las Vegas, NV), October 26–29 2009.

[17] S. Kopparty, S. Krishnamurthy, M. Faloutsos, and S. Tripathi, "Split TCP for mobile ad hoc networks," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, (Taipei, Taiwan), pp. 138–142, November 2002.

[18] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 27, pp. 19–43, Oct. 1997.

[19] T. Goff, J. Moronski, D. Phatak, and V. Gupta, "Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments," *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3, pp. 1537–1545 vol.3, March 26–30 2000.

[20] "Debian linux distribution." http://www.debian.org/, June 2012.

[21] "Quagga routing suite." http://www.quagga.net/, December 2009.

[22] J. Moy, "OSPF Version 2." RFC 2328 (Standard), Apr. 1998. Updated by RFC 5709.

[23] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "OSPF for IPv6." RFC 5340 (Proposed Standard), July 2008.

[24] G. Malkin, "RIP Version 2." RFC 2453 (Standard), Nov. 1998. Updated by RFC 4822.

[25] G. Malkin and R. Minnear, "RIPng for IPv6." RFC 2080 (Proposed Standard), Jan. 1997.

[26] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)." RFC 4271 (Draft Standard), Jan. 2006.