The Click Convergence Layer: Putting a Modular Router Under DTN2

Agoston Petz, The University of Texas at Austin

Brenton Walker, Laboratory for Telecommunications Sciences

Calvin Ardi, Laboratory for Telecommunications Sciences

Christine Julien, The University of Texas at Austin

TR-UTEDGE-2010-016

The Click Convergence Layer: Putting a Modular Router Under DTN2

Agoston Petz
University of Texas at Austin
agoston@mail.utexas.edu

Brenton Walker
Laboratory for
Telecommunications Sciences
brenton@ltsnet.net

Calvin Ardi
Laboratory for
Telecommunications Sciences
calvin@ltsnet.net

Christine Julien
University of Texas at Austin
c.julien@mail.utexas.edu

ABSTRACT

The Bundle Protocol shows great promise as a general purpose application-layer protocol for delay-tolerant networks (DTNs) and has found many adopters within the research community. As an application layer protocol, a convergence layer (UDP/TCP sockets or some other domain-specific transport mechanism) is required to deliver bundles between nodes. We argue that a flexible convergence layer architecture that is easy to use and modify is beneficial to researchers since it enables them more easily experiment with new protocols for DTNs. We introduce such an architecture incorporating the Click Modular Router as a new highly configurable and extensible convergence layer, provide an implementation for a Click convergence layer adapter in the popular DTN2 Bundle Protocol reference implementation, and present a performance comparison between the Click convergence layer and native DTN2 convergence layers.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Store and forward networks

General Terms

Design, Experimentation, Performance

Keywords

Delay-Tolerant Networks, Bundle Protocol, Convergence Layer, Architecture, Click Modular Router

1. INTRODUCTION

The bundle protocol [12] is an application-layer overlay protocol designed to support communication in delay-tolerant networks (DTNs) in which nodes may experience intermittent connectivity, high delay links, or nodal churn. Besides working in these challenged environments, the bundle protocol (BP) provides a generalized model for communication via arbitrary data units called "bundles". One advantage of the bundle protocol is that it abstracts away the network stack and choice of network protocols from the application developer, and instead presents a unified API suitable for use in a variety of environments. This makes the bundle protocol an ideal vehicle for the development, integration, and testing of alternative transport, network, and link layer protocols. The key feature of the bundle protocol that makes this possible is the modular interface to one or more "convergence layers" which act as interfaces between bundles and the network stack. As defined in RFC5050 [12], convergence layers have two responsibilities: to send and receive bundles on behalf of a Bundle Protocol Agent (BPA).

The prototypical convergence layer, and probably the most widely used for terrestrial applications, is the TCP convergence layer (TCPCL). It takes advantage of the connection-oriented nature of TCP, as well as its reliable transport and wide availability to send and receive bundles in a variety of environments. Other convergence layers that have been proposed or implemented to some degree include the UDP convergence layer (UDPCL) which encapsulates bundles in UDP datagrams, the Ethernet convergence layer which transmits bundles encapsulated in raw Ethernet frames. Convergence layers have also been developed for protocols specific to delay-tolerant networks, including the Licklider Transmission Protocol (LTP) [10], the Nack-Oriented Reliable Multicast protocol (NORM) [4], Saratoga [14], a protocol for space communications, and AX.25 [11] for amateur packet radio.

Convergence layers often provide services to the BPA in addition to sending and receiving bundles. For example, the TCPCL provides partial delivery information to the BPA in order to facilitate reactive fragmentation of large bundles. Several of the convergence layers mentioned above provide some sort of reliability or connection status notification. These additional services make the interface between the BPA and the Convergence Layer Adapter (CLA) more complicated than it initially appears to be, often blurring the layers of cross-layer protocol design. Researchers developing new convergence layers face two daunting challenges: implementing new low-layer protocols, which may involve writing elaborate and highly OS-specific code and kernel-level debugging, and integrating those protocols with the somewhat complex convergence layer interface architecture

ExtremeCom 2010 Dharamsala, India

of DTN2 [1], the open-source bundle protocol reference implementation. The Click Modular Router [7] is a tool that can simplify the first problem. By building a CLA interface between Click and DTN2, we aim to simplify the second problem.

1.1 The Click Modular Router

The Click Modular Router [7] is a flexible, modular, and stable framework for developing routers. Click has a large base of available network protocols and is itself a good framework for implementing experimental protocols. Click is written in C/C++, runs on Linux and BSD, and includes numerous components to manipulate the network stack from the hardware drivers to the transport layer. This frees network developers from reimplementing common algorithms and network stack constructs. Click's modular nature offers the flexibility of easily swapping in alternative components or modifying existing protocol components, and since it runs as a stand-alone user process, new network stacks can be tested and debugged without breaking the operating system's network layer. If additional performance is required, Click can be compiled into Linux kernel modules and run directly in the kernel, thus replacing the native network protocol implementations.

Every instance of Click consists of a given set of elements that are "wired" together by configuration files to define a working protocol stack, where every element has explicitly defined input and output ports that must be connected. The order in which these elements are wired together defines the functionality of that particular instance of Click. Such configurations within Click can be as simple as a single element such as `Print`, which prints out the contents of every packet that passes through it, or as complicated as an entire working implementation of the TCP/IP stack, complete with ARP tables, routing, and so on.

2. PROBLEM STATEMENT

Our objective is to build a Convergence Layer for Click which enables researchers to implement new convergence layer protocols, or to modify existing ones, simply by manipulating Click elements in the Click modular router framework. This will simplify the process of deploying new convergence layers for the bundle protocol. Prior to this work, each new convergence layer had to be coded directly into the bundle protocol agent or linked from a convergence layer adapter in the BPA. In practice this meant that any new convergence layer had to be designed and built to work directly with the available APIs and device driver libraries in the Linux kernel. This is often a complex task which we strive to greatly simply with our architecture. We aim to create a generic Convergence Layer Adapter (CLA) for the Click Modular Router with which any new convergence layer can be designed for and built directly in Click.

3. RELATED WORK

The External Convergence Layer (ECL) [3], developed by BBN Technologies as part of an earlier phase of the SPINDLE [6] project is similar in spirit to our work. ECL was designed to use an XML interface to enable the BPA to interact with convergence layers implemented as external applications. In the ECL, message passing is done through a local TCP socket and bundles are written to disk in order to facilitate copying them between the BPA and the convergence layer implementation. This usage of the disk, in addition to the requirement of serializing the bundle protocol data into an XML format, introduces unnecessary overhead. The bundles as generated by the BPA are in a form that is readily copied into packet buffers and sent down the network protocol stack. Though we appreciate the value of a generic format such as XML, in our case it was not optimal to deconstruct a bundle into XML format, just to reconstruct it back to a continuous data buffer so it can be processed by the network stack.

Recently, the External Convergence Layer has essentially been deprecated due to the lack of usage in the DTN community and BBN Technologies' move to implement SPINDLE3, a proprietary BPA with interfaces and CLAs incompatible with those of the DTN2 bundle protocol reference implementation.

4. DESIGN AND IMPLEMENTATION

Designing the Click convergence layer (ClickCL) was challenging for a number of reasons. First, we needed to understand the interface between the Bundle Protocol Agent (BPA) and its convergence layers, which in practice is not as clean as RFC5050 would lead one to believe. Additionally, unlike monolithic convergence layers which exist entirely within their respective BPA, ours must interface the BPA to a separate process (since Click runs as a standalone process). Therefore we needed to design a protocol to interface the BPA and Click by means of IPC. We also needed to split the functionality of the convergence layer between the Click Convergence Layer Adapter (ClickCLA) which runs in the BPA and Click itself.

Another fundamental problem was deciding what services to include in the interface between the BPA and Click. The formal requirements put on any convergence layer adapter regarding the sending and receiving of bundles are very open-ended, and many convergence layers provide additional services to the BPA. These additional features lead to more elaborate integration of the CLA with the BPA. Since we wanted to ensure that any conceivable convergence layer could be built using the ClickCL, we needed to design an open-ended interface that could be easily extended. To accomplish this we designed a control protocol to carry such metadata between Click and the BPA. We will consider ourselves successful if our BPA/CLA interface provides enough flexibility to implement any of the existing CLs as ClickCL modules. To demonstrate the viability of this concept we implemented an example convergence layer using the ClickCL, the details of which are covered in the next section.

4.1 Integration with DTN2

We implemented our Click convergence layer for the DTN2 bundle protocol reference implementation [1]. DTN2 implements the BPA as a multi-threaded daemon with separate convergence layer adapters for each available convergence layer. Most of the existing convergence layer adapters (like the TCP and UDP CLAs) rely on user-level libraries to interface with the network stack. For example, the UDP and TCP CLAs both call into the popular Linux Sockets API to transmit and receive bundles from the network. Thus, the Linux kernel's network stack provides the mechanisms that process bundles into datagrams or streams, respectively.

Click does not provide such a well-documented API to

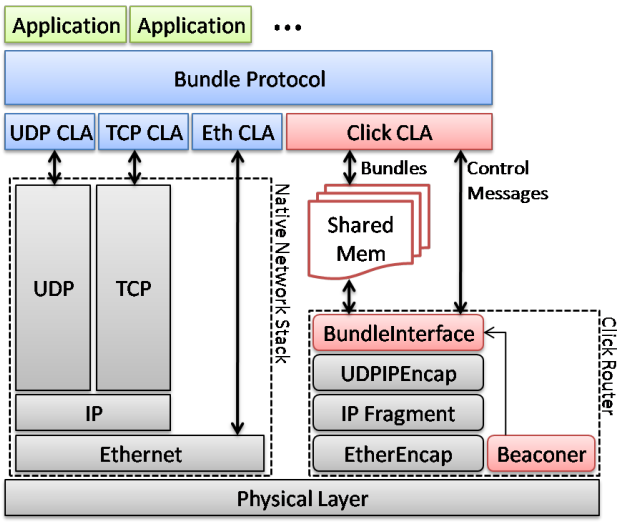


Figure 1: Architecture of the Click Convergence Layer

external applications, so the ClickCLA cannot call Click's functions directly. Instead, the ClickCLA must transfer the bundles to Click by other means of inter-process communication. We designate two separate channels to transfer bundles between the bundle daemon and Click: a control channel which carries control messages and information about incoming and outgoing bundles, and a shared memory "channel" for the transfer the actual bundles. We chose POSIX shared memory mapping to transmit bundles since they can be up to 2GB in size according to the bundle protocol specification, and it is the most efficient way to handle such large data chunks. This design, shown in Figure 1, puts an extra burden on our Click module to recognize the bundle format and primary bundle block header. DTN2 provides two calls, `produce()` and `consume()`, to receive and send bundles in increments appropriately chosen by the convergence layer module without the convergence layer having to worry about the format or content of the bundle. In the near future we will expose this functionality to the ClickCL through our shared memory/control message communication framework.

4.2 The Click Network Stack

The network stack for the Click convergence layer is implemented as a single instance of the Click Modular Router. For the purposes of testing against an existing DTN2 convergence layer, we constructed a UDP stack entirely in Click to compare against DTN2's internal UDP Convergence Layer (UDPCL). Figure 1 depicts our stack configuration for this specific ClickCL module. The grey elements in the figure (`UDPIPEncap`, `IP Fragment`, and `EtherEncap`) are all elements available in the default Click installation. Together, they package all incoming packets into a UDP datagram and send it out via the wireless interface. In addition to using these elements, we have added the `BundleInterface`, which passes bundles and control messages to and from the bundle daemon, and the `Beaconer`, which provides neighbor discovery through the use of beacons from which a table of currently connected nodes is built. This beaconing feature is not provided by the native UDPCL.

4.3 The Click Convergence Layer Adapter

A Convergence Layer Adapter passes outgoing bundles and receives incoming bundles onto and from the network, respectively. ClickCLA is the BPA's interface to Click and provides the functionality of a generic convergence layer. We defined a custom control protocol to communicate between the ClickCLA and the `BundleInterface`. The interface currently supports four types of control messages: BUNDLE_READY, BUNDLE_SENT, LINK_UP, and LINK_DOWN. For our initial implementation, we chose to use the Linux universal TUN/TAP interface to transmit the control messages between the `BundleInterface` and the ClickCLA in the bundle daemon. As discussed in Section 5, this may not have been the optimal choice, but our current design provides us with the desired functionality. To illustrate the functionality of the Click Convergence Layer, we describe the process of sending one bundle between two nodes **A** and **B**. The example assumes that DTN2 is configured to add next-hop routes, though static routes could also be used.

- **A** generates a bundle destined for **B**, but since they are not connected, the bundle gets queued by the bundle daemon
- **B** moves in range and **A**'s `Beaconer` discovers **B**
- The `Beaconer` notifies the `BundleInterface` of a new link
- The `BundleInterface` generates a LINK_UP control message and sends it to ClickCLA
- The ClickCLA creates a link in the bundle daemon for node B and adds a next-hop route to the bundle daemon's routing table
- **A** sends the bundle destined for **B** out on the newly available link through the ClickCLA
- The ClickCLA copies the bundle into a shared memory block and sends the block's identifier in a BUNDLE_READY control message to the `BundleInterface`
- The `BundleInterface` receives the BUNDLE_READY message, processes the bundle into a Click packet, and passes it down to the `UDPIPEncap`, which encapsulates the bundle in a UDP frame, and again in an IP frame
- The `IPFragmenter` fragments the packet into MTU-sized chunks, adding its own headers and passes it to the `EtherEncap` element to encapsulate the fragments into an Ethernet frame
- `EtherEncap` passes the ready packets to the wireless driver which sends it on the network

The IP fragments are then reassembled, the headers are stripped, and the bundle is delivered by **B**'s Click instance to **B**'s bundle daemon in a similar fashion. When the two nodes part ways, the `Beaconer` times out their connection and LINK_DOWN messages are generated by both parties to disable the links and remove the routes. For the purposes of sending and receiving bundles, the functionality of

our Click-based UDP convergence layer is identical to that of the native UDP convergence layer, and thus they are interoperable with one another. Additionally, the Click-based UDPCL is is completely modular and easy to modify, add, or remove functionality from. In contrast, one would have to delve into UDP implementation within the Linux kernel to modify the native UDPCL. Changing any Click-based convergence layer requires only simple manipulation of existing or inclusion of new custom elements in the processing stream. In most cases, no changes are required to the convergence layer adapter in the bundle daemon unless new control message types were added, as each additional type requires its own handler in the CLA. We did this ourselves in order to remove the clunky application-level beaconing that the bundle daemon was doing for node discovery, and replace it with a more compact beaconing method in Click. Many new convergence layers could be defined without changing anything in the CLA.

4.4 Routing

In our current implementation, bundle forwarding decisions are made exclusively in the routing module of the BPA. Though Click is a "modular router", it does not participate in any bundle forwarding decisions and simply functions as an alternate network stack. In the future, we plan to add a routing interface to the bundle daemon to allow routing decisions to be made by Click. DTN2 includes an external router interface [2] that has been used to implement functional routing modules [5]. By extending our Click framework to interact with DTN2's external router interface, we will be able to implement more cross-layer context-aware routing algorithms. Since the Click Modular Router was designed to facilitate the implementation of routing algorithms, it already has much of the functionality common to all routing protocols implemented in modular elements. This further integration will enable code reuse and expand the possibilities for new DTN routing algorithm implementation.

5. EVALUATION

To evaluate our Click convergence layer, we sought to do a performance test against a native convergence layer. We decided to build a UDP convergence layer in Click that is analogous to the UDP convergence layer in the bundle daemon, and to test the performance of each in both a wireless and wired environment.

5.1 Wireless Experiments

Our wireless experimental setup consists of two nodes with VIA NR10kEG nano-ITX motherboards, 1GHz VIA C7 processors, and 1GB DDR2 RAM each equipped with Atheros 802.11bg wireless cards. The computers themselves are identical to the computers on the Proteus [9] nodes of the Pharos Mobile Computing Testbed [13]. In all wireless tests, the nodes were placed 10 feet apart in an indoor lab space.

We used the **dtnsource** and **dtnsink** applications to generate fixed size bundles at regular intervals and record bundle receptions at the destination. Both programs are distributed with DTN2. Though the UDPCL encapsulates bundles in UDP datagrams, which have a maximum size of 65kB, we found that using anything larger than 48kB bundles resulted in IP fragmentation problems at the sink, and a throughput of close to 0. Therefore we used 48kB bundles in all of our tests.

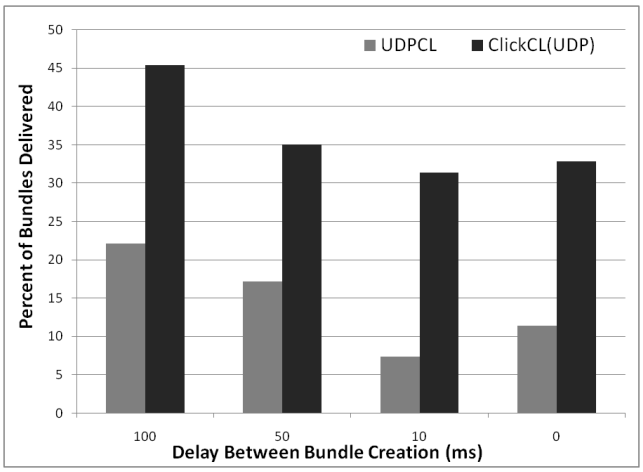


Figure 2: Bundle Delivery Ratios

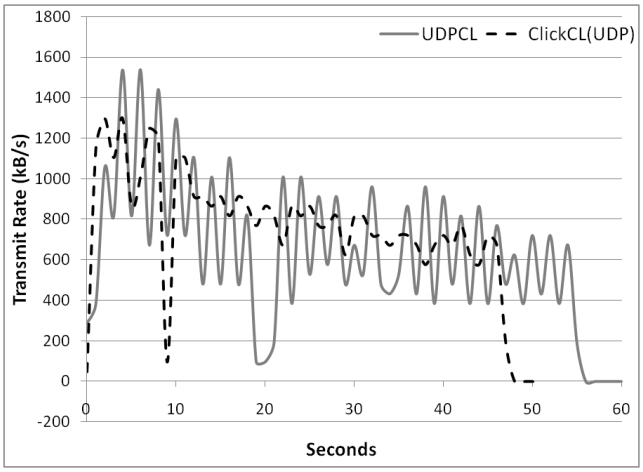


Figure 3: Transmission Rates in kB/s

We ran several tests with the source creating bundles with 100ms, per 50ms, per 10ms, and 0ms pauses between bundles. Figure 2 shows the delivery ratios for one set of such tests. Our results were consistent between different tests. In all cases, the Click convergence layer implementing UDP, denoted ClickCL(UDP) in the graphs, delivered a larger percentage of the bundles than the native UDP convergence layer. We sought to understand why this might the case, since the performance should have been nearly identical. We discovered that the ClickCL(UDP) has a much less bursty transmission rate at the wireless card than the native UDPCL, which we suspect contributes to the successful delivery and reassembly of more UDP datagrams.

Figure 3 shows the transmission rates at the wireless card for an experiment in which dtnsource was not throttled. We observed this phenomenon in all of the tests, but it was especially evident when the bundle creation rate was not throttled. We do not plot the receiver's data rate since it is nearly identical the transmitter's. We hypothesize that Click's smoother transmission rate is a side-effect of the extra processing it must do over the native UDP implementation of the kernel and the polling nature of the Click wireless

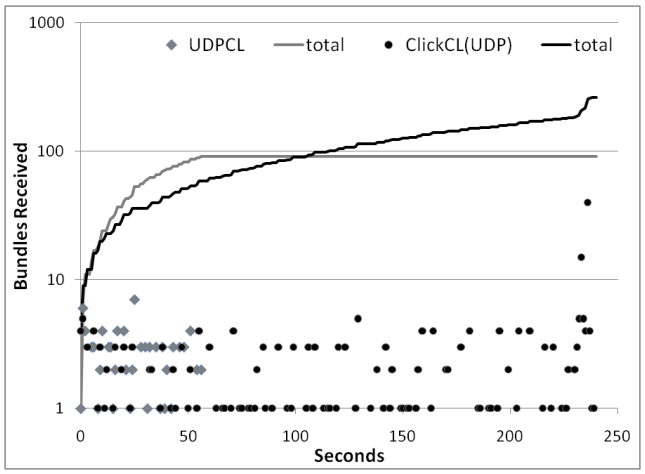


Figure 4: Bundle Delivery Latencies (800 48kB Bundles)

device driver interface. This seems to have a positive effect on packet delivery ratios when using unreliable protocols over lossy channels. We do not claim this as a contribution of our work, but it is an interesting result nonetheless.

We also used dtnsink to study bundle delivery latency. Figure 4 shows the number of bundles received by dtnsink every second for the same test as Figure 3. The diamonds and circles indicate the number of bundles delivered for UDPCL and ClickCL(UDP) respectively, and the lines indicate the total number of bundles received up to that point in the test. Note the logarithmic y-axis. This graph shows that DTN2 with UDPCL delivers bundles to dtnsink about as fast as the source can send them, with little delay. The ClickCL(UDP) has a much longer latency for processing incoming bundles and delivering them to the dtnsink application. In fact, as you can see in Figure 3, the transmission of packets (and the receipt of them at the receiver's wireless card) is finished approximately 60 seconds into the test, whereas ClickCL(UDP) continues to deliver bundles to dtnsink until around 250 seconds.

We discovered that incoming bundles, although processed very quickly by Click, and copied very quickly to shared memory, were experiencing long delays because of the design of the ClickCLA in the bundle daemon, which spent a lot of time working through the queue of control packets to process all of the available bundles. We suspect that this is due mostly to our decision to encapsulate control packets into Ethernet frames and pass them between the ClickCLA and the `BundleInterface` in click via the TUN/TAP device in Linux. Though this seemed like a simple way to accomplish the exchange of control packets, and allowed us to reuse existing code from other DTN2 convergence layers, it introduces unnecessary delays in the processing of incoming bundle data while the frames are delivered by the TUN/TAP device and processed by the ClickCLA. In our next iteration of the ClickCLA, we will choose a more efficient IPC mechanism such as message passing.

5.2 Wired Experiments

Our wired experiments were designed to test the maximum performance of DTN2 with the two UDP convergence

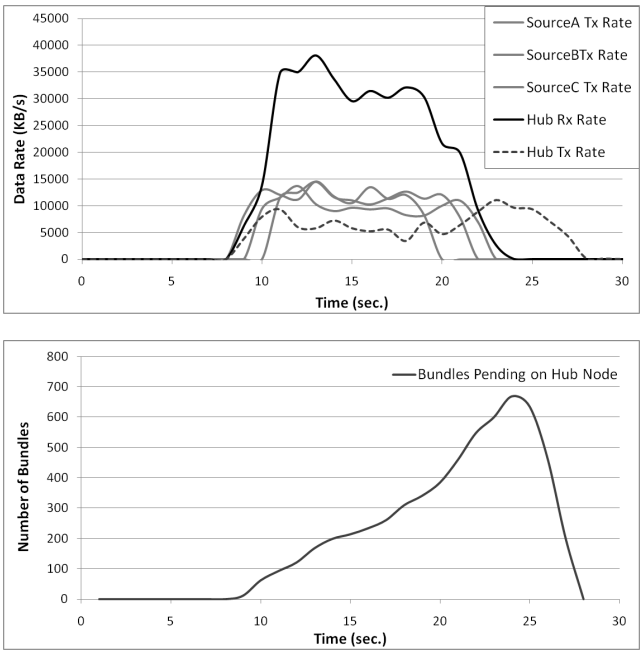
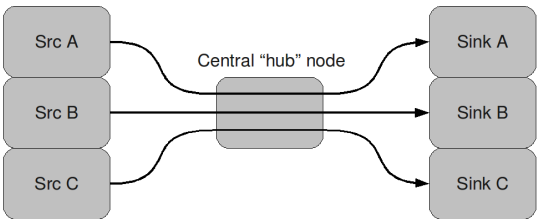


Figure 5: Three Flow DTN2 Performance Evaluation on Gigabit Ethernet

layers. We had reason to believe that the act of sending and receiving bundles through the API incurs a non-trivial amount of overhead, so we designed the experiment to isolate the performance of the CL independent of the API. Our results led to some interesting discoveries about what happens when convergence layers are overloaded.

Our experimental setup consisted of seven nodes: three sources, three sinks, and one central "choke-point" or "hub" node as illustrated in below.



Since the central hub node does not have to pass bundles through the DTN2 API, its performance is only a function of the CL's ability to send and receive bundles, and the BPA's ability to process them. Because it handles three unthrottled flows of bundles at once, we are sure to test the maximum performance of the stack. In this experiment we used Dell Studio Hybrids with Intel T8100 2.1 GHz CPUs connected to a single Gigabit Ethernet switch. We used the **dtnsource** and **dtnsink** applications to generate and dispose of bundles. In this experiment, each source generated 2000 48kB bundles destined for its sink.

Figure 5 illustrates the performance of the native DTN2 UDPCL in a typical experiment. The upper plot shows the transmit rate of the three source nodes and the receive and transmit rates of the hub node. We immediately notice that

the hub node is receiving bundles at a much higher rate than it is transmitting them. This leads to a backlog of bundles on the hub node. We also notice that the hub's transmit rate is higher when it is done receiving bundles from the sources. What is surprising is that the hub node's pending bundles list continues to grow for at least several seconds after the sources stop transmitting. This indicates that a considerable backlog of incoming data is queued by the UDP stack, and this data is processed as incoming bundles and passed to the BPA at its own pace.

Running the same test using the ClickCL(UDP) on all the nodes, we observed about a 4x degradation in throughput for these high-speed wired tests. As observed in our wireless tests, the cause seems to be that when the stack is heavily loaded, our control messages between the Click CLA and Click are delayed considerably. Modifying this IPC mechanism is a straightforward task that will drastically improve the ClickCL performance under high-speed loads.

6. FUTURE WORK AND CONCLUSION

We have introduced the Click convergence layer architecture, and our first implementation of it for the DTN2 reference implementation of the bundle protocol. We have also shown that a UDP convergence layer, although not novel, can be easily constructed with just a few Click elements, and that our Click convergence layer has comparable performance to the native convergence layers available in DTN2. However, the benefits of a Click-based convergence layer go far beyond the implementation of new network protocols for delay-tolerant networks. Click can easily become a router, and is a great framework for the development and implementation of routing algorithms. We intend to build a routing interface to DTN2 to enable Click to be used as a bundle router in addition to a convergence layer, thus enabling developers of routing algorithms for delay-tolerant networks to implement their ideas easily and directly with reusable Click elements instead of coding them into the DTN2 reference implementation itself. This has the added benefit of placing the routing algorithms closer to the physical and link layers, thus fascinating a much faster reaction time to changes in the network. In its current iteration, DTN2 implements routing algorithms at the application layer. In a highly dynamic environment, very brief communication opportunities could be easily missed by the slow reaction time of such an implementation.

Furthermore, using Click allows us to leverage prior work in context aware adaptation to delay-tolerant environments [8]. The context gathering ideas developed in [8] could be extended to provide information about the state of the network to the bundle daemon. Additionally, the adaptive Click-based networking stack could be used as a convergence layer for the bundle protocol by adding our CLA.

7. ACKNOWLEDGEMENTS

The authors would like to thank Vidur Bhargava of UT-Austin, Matt Beecher of LTS, and Ginnah Lee of the University of Maryland for their help. The authors would also like to thank the Center for Excellence in Distributed Global Environments, the Laboratory for Telecommunications Sciences, and the University of Maryland Institute for Advanced Computer Studies for providing research facilities, personnel, and a collaborative environment. This research was funded in part by the US Department of Defense. The views expressed are those of the authors, and do not necessarily reflect the views of the sponsoring agencies.

8. REFERENCES

[1] Bundle protocol reference implementation. `http://www.dtnrg.org/wiki/Code`.

[2] DTN external router interface. `http://www.dtnrg.org/docs/code/DTN2/doc/external-router-interface.html`.

[3] DTN2 manual: External convergence layer. `http://www.dtnrg.org/docs/code/DTN2/doc/manual/cl-extcl.html`.

[4] B. Adamson, C. Bormann, M. Handley, and J. Macker. Negative-acknowledgment (NACK)-oriented reliable multicast (NORM) protocol. *Internet Society Request for Comments RFC*, 3940, 2004.

[5] A. Balasubramanian, B. Levine, and A. Venkataramani. DTN routing as a resource allocation problem. *SIGCOMM Comput. Commun. Rev.*, 37(4):373–384, 2007.

[6] R. K. *et al.* The SPINDLE Disruption-Tolerant Networking System. In *MILCOM*, 2007.

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[8] A. Petz, A. Bednarczyk, N. Paine, D. Stovall, and C. Julien. MaDMAN: A middleware for delay-tolerant mobile ad-hoc networks. Technical Report TR-UTEDGE-2010-010, 2010.

[9] http//proteus.ece.utexas.edu.

[10] M. Ramadas, S. Burleigh, and S. Farrell. Licklider transmission protocol–specification, IETF RFC 5326. `http://www.ietf.org/rfc/rfc5050.txt`, 2007.

[11] J. Ronan, K. Walsh, and D. Long. Experiments with Delay and Disruption Tolerant Networking in AX. 25 and D-Star Networks. 2009.

[12] K. Scott and S. Burleigh. Bundle protocol specification, IETF RFC: 5050. `http://www.ietf.org/rfc/rfc5050.txt`, November 2007.

[13] D. Stovall, N. Paine, A. Petz, J. Enderle, C. Julien, and S. Vishwanath. Pharos: An application-oriented testbed for heterogeneous wireless networking environments. Technical Report TR-UTEDGE-2009-006, The Center for Excellence in Distributed Global Environments, The University of Texas at Austin, 2009.

[14] L. Wood, W. Eddy, W. Ivancic, J. McKim, and C. Jackson. Saratoga: a Delay-Tolerant Networking convergence layer with efficient link utilization. In *IWSSCŠ07*, 2007.