Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2011 (WowKiVS 2011)

EZgate - A flexible Gateway for the Internet of Things

Torsten Teubler, Ulrich Walther, Horst Hellbrück

12 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

EZgate - A flexible Gateway for the Internet of Things

Torsten Teubler[1], Ulrich Walther[2], Horst Hellbrück[1]

Lübeck University of Applied Sciences[1], fluid Operations [2]

Abstract: Two years ago a survey of the wireless world research forum predicted that in the year 2017 there will be seven trillion wireless devices for seven billion humans which is equivalent to 1000 devices per human being on the average. The future will show if this incredible number will be reached but for sure we will see an increasing number of wireless devices forming the Internet of the future. The new evolving "Internet of Things" is one of the challenging research topics today. With many wireless resource constraints devices, smart gateways integrating these small battery-powered devices into the future Internet will play a major role for the success of the Internet of Things. These gateways will work as a communication endpoint or proxy enabling transparent services including mechanisms for semantic service discovery, Quality of Service (QoS), and performance enhancing proxies (PEPs). In this work we will introduce a fully operable TCP/IP-Stack EZgate written in Java that allows designing and implementing such gateways for wireless networks in a flexible and fast approach and compare it with related work. We will demonstrate how the protocols in this stack can be assembled in a flexible manner, creating various types of gateways and can be easily extended to implement cross layer techniques. Finally, we evaluate the performance of the implementation for delay and throughput performance to show that EZgate is suitable for use in a productive environment.

Keywords: Userspace Protocol Stack, Cross Layer Gateway, Wireless Sensor Networks, Internet of Things, Ad-Hoc Network

1 Introduction

Today new devices and applications continuously enrich and thereby enlarge the Internet. The Future Internet comprises those new devices and cutting-edge technologies today in this development are smart phones. After a difficult start with various setbacks like low bandwidth compared to the high costs and short battery life time, today they are well established and are one of the fastest growing sections of the Internet. The next large challenge for the established Internet technologies are the integration of Ad-Hoc networks like mobile Ad-Hoc networks (MANETs) or wireless sensor networks (WSN) with a massive number of devices with reduced capabilities and their data traffic [Dav08, GCM+08]. The impact of those emerging technologies is currently described with terms like "Internet of Things" or "Smart Objects" and is seen as THE change in networks. Additionally, new technologies like network virtualization [CB10] and data centric networking [JMSG07] emerge in recent research.

In the near future IPv6 will play a bigger role in the Internet not only because of dwindling IPv4 address space. Although IPv6 discloses many challenges its deployment is delayed for ten

years after its invention. For new networks like MANETs of WSNs IPv6 is the protocol of choice at the moment. However, that might change in future. As it is not clear if IPv6 will displace IPv4 complete they might coexist side by side for many years. For the latter case protocol conversion is essential for a smooth operation. Additionally higher level protocols are affected of continuous changes. For example SCTP [Ste07] is a competitor to TCP and combines the advantages of TCP and UDP. A protocol conversion is essential in these cases too if we require a smooth transition with both solutions in place.

The situation is comparable to the NAT (Network Address Translation)-debate in the mid-90s. At that time the deployment of NAT-Traversal on special gateways solved the address shortage of IPv4 and thereby enabled the evolution of the Internet as we know it today. With that history and future development in mind flexible gateways will become a vital part of the Internet and will facilitate further evolution of the Internet.

As a result the future Internet (of Things) will become a colorful bunch of different technologies owing through emerging new devices and applications. The challenge of this progress is that devices – especially those at the heart of the Internet – have to handle different protocols and algorithms at *all* layers. The tasks in future networks will include deployment of new technologies. One of the problems is to avoid changing hardware and performing neither tedious development tasks nor adjustments to software and firmware which is closely tied to the devices hardware. Gateways can support these new technologies and as a proxy enhance the performance (PEP) of the system reducing the load from the small battery powered devices in the Internet of Things.

In this work we present a suitable Java based approach enabling Internet devices with gateway functionality to be ready for the aforementioned future tasks. The suggested solution allows programming in the user space resulting in a flexible plugging of protocols together tailored to the needs of the tasks and allowing cross-layer techniques at all layers. With the potency of Java including various libraries for handling XML and implement Web services, semantic relevant functionality can be easily implemented in the future.

The rest of the paper is organized as follows. The next section introduces related work and discusses the need for a more flexible approach beyond the existing solutions. Section 3 introduces the basic architecture of the Java stack. Section 4 discusses the details of the flexible gateway adaptations and illustrates the intuitive extension of the stack functionality. Section 6 provides first evaluation results. We will conclude this work in Section 7 and give a future outlook.

2 Related Work

This section is structured in two parts. First we will introduce a Java based approach for user space networking stack and also other user space network stack approaches will be briefly mentioned. In the second part we will introduce a gateway approach which connects a wireless sensor node to the Internet.

Our work is a continuation of EZnet [WF02] which was introduced as a framework for rapid prototyping of protocols. We substantially extended EZnet and updated it for TCP and UDP in IPv6 and implemented the gateway functionality. Furthermore, we redesigned and optimized EZnet for usage in a productive environment beyond its original target field which was rapid prototyping.

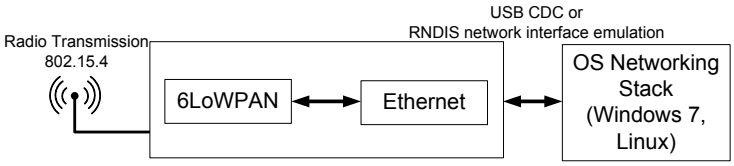


Figure 1: Architecture of the "Jackdaw" Raven USB stick

[SC05] introduces a user space development of Netfilter[1] protocol implementations. The complete software development in C and debugging is performed in the user space. In the next step the protocol implementations are translated into kernel modules and executed in the kernel. This approach requires that the developer is familiar with the Linux kernel API and debugging with GDB or DDD is not yet supported.

Further work in the area of user space networking stack is presented in [ESW01, EM95]. The aforementioned approaches are restricted to a special operating system or target prototyping of protocols and research and are not used in a productive environment. They all focus at layers below the application layer. Additionally we support cross layer interaction among layers including application layer. We see this as a competitive advantage when implementing PEP, QoS mechanisms or semantic service discovery. In addition the application area for our approach is quite different. We consider the connection of resource constrained devices to the Internet with our gateway and we expect less traffic like normal Internet routers are exposed to. To the best of our knowledge there is no approach which combines the advantages of a user space networking stack with a cross layer enabled gateway.

A solution for a gateway between wireless sensor nodes and the Internet is the "Jackdaw"[2] Raven[3] USB stick. It is part of the Contiki operating system [DGV04] and appears as an IPv6 network interface card on the off-the-shelf PC while plugged in. The USB stick together with the driver converts Ethernet frames from the PC to IEEE802.15.4/6LoWPAN [MKHC07] frames and vice versa. The Jackdaw stick has the potential for a valuable part of a gateway as it provides switching capabilities between different technologies in the data link layer (IEEE 802.15, IEEE 802.3). It can serve as the base for connecting sensor nodes running IPv6 to the Internet. Fig. 1 illustrates the approach. The Raven stick depicted as rectangle in the middle is attached via USB CDC[4] or RNDIS[5] to a Linux or Windows driven PC on the right. The left side of the image shows the antenna that sends out 802.15.4/6LoWPAN frames.

As illustrated in Fig. 1 the stick converts protocols between 6LoWPAN and Ethernet. This is due to the operating systems TCP/IP Stack lacks handling 6LoWPAN packets so they are translated to Ethernet frames. As the stick appears as a standard Ethernet network interface via CDC or RNDIS technology the packets are processed by the OS stack like standard Ethernet frames from a wired link.

One drawback is that the code for the stick is written in highly optimized C which is hard to

[1] Linux kernel hook handling framework for intercepting/manipulating network packets
[2] http://www.sics.se/~adam/contiki/docs/a01462.html
[3] http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=4291
[4] http://www.usb.org/developers/devclass_docs/CDC1.2_WMC1.1.zip
[5] http://www.microsoft.com/whdc/device/network/NDIS/rmNDIS.mspx

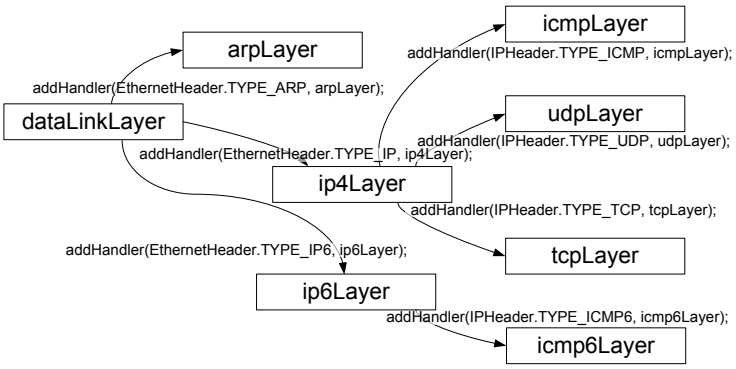


Figure 2: Network layers independently linked in EZnet

maintain and exchange in case of further adoptions. However, this is a crucial point because the 6LoWPAN standard recently changed often and we can predict that future Internet technologies and algorithms are changing quickly during introduction as well. Another drawback is that the protocol conversion is completely done on the resource constrained microcontroller driven Raven stick.

In our perception it is a tedious task to change device firmware. Consequently, we suggest a more flexible approach. Nevertheless the Raven stick can be successfully integrated in any gateway software and is also integrated into our work.

3 EZnet Architecture

When the initial work on the EZnet framework was started nearly a decade ago, the primary goal was to provide an easy way for building prototypes of new network functionality, independent of the network layers that need enhancements or extensions [WF02]. This is even more important today, where the uprise of mobile devices marks a new era of personal computing. Since most of the networking functionality of mobile operating systems is placed in kernel space, ”quickly” adding new features or functionality doesn't really work, as development in kernel space usually requires ten times the effort when compared to user space.

Based on these design goals, the EZnet framework has the following properties: user space, provides entire TCP/IP stack, clean architecture, well documented, performance, enables use in real world scenarios, ability to plug in simulators. The language of choice was Java, because it is well-known, object oriented, and because of the good availability of IDEs and tools. EZnet properly models the layers of the Internet Protocol Suite and implements the Ethernet Layer, IP (4 and 6), ARP, ICMP, IGMP, UDP, TCP, and DHCP. All network layers inherit from the Java class ProtocolLayer. They can be linked together independently with the `addHandler` method as depicted in Fig. 2.

Each network packet's payload is encapsulated by a Packet, and each layer optionally adds a PacketHeader. Packet headers are also used to represent trailers used in some protocols. One of the most important tasks is the marshalling and unmarshalling of network packets of a specific protocol, which happens in the protocol's specific implementation of PacketHeader. An imple-

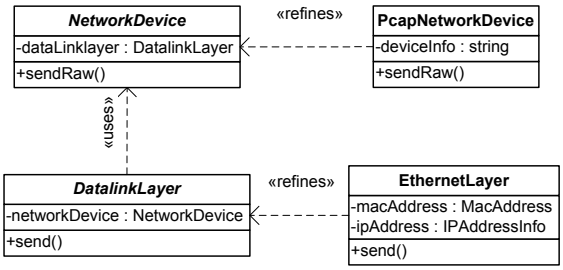


Figure 3: UML diagram showing the relation between DatalinkLayer, NetworkDevice and their refinements

mentor can either decide to implement marshalling/unmarshalling manually, which might be a good choice if the protocol is relatively simple, or might choose to use one of the automated code generation mechanisms to generate marshalling/unmarshalling code from XML specification, or from ASCII tables as they are frequently used in Requests for Comments (RFCs). In order to achieve a tolerable performance, the payload is actually never copied (zero copy strategy) and processing happens in place wherever appropriate.

When a packet is received by the DataLink layer it is copied to a ByteBuffer and further processed by the chain of protocol layers. Each layer first unmarshals its protocol header, which is initiated by the protocol stack that calls the header's `disassemble` method. After unmarshalling the header the layer determines if the protocol's payload needs to be passed up to a higher level protocol, in which case the same procedure is repeated for the next higher protocol layer.

If there is no higher protocol layer, the protocol layer's `process` method is called which actually processes the received payload. For UDP and TCP, the layer passes the payload to the application and triggers signals to the API.

4 EZgate Approach

EZnet formerly was designed with access to a single network and thus only contains one MAC address and a single IP address. The latter point is an artificial constraint that needs to be removed in a productive environment as standard PC operating systems allow several IP addresses for one NIC. We modified this in our current implementation, where IP addresses are attributes of the class EthernetLayer as IP addresses are associated with a network interface. The stack is extended now to support more than one DatalinkLayer instance which is the lowest layer and responsible for handling frames of a device driver. EthernetLayer is a refinement of the DatalinkLayer which is the abstract base class. During the stack initialization the DatalinkLayer receives a reference pointer to a NetworkDevice object. NetworkDevice is an abstract class and acts as an adapter to a real network device or interface. We implemented the PcapNetworkDevice for first tests that sends and receives Ethernet frames via Libpcap to a real network adapter on the PC. The relation between DatalinkLayer, EthernetLayer, NetworkDevice and and PcapNetworkDevice is depicted in Fig. 3.

However, the support for several network interfaces is only the first step for building a flexible

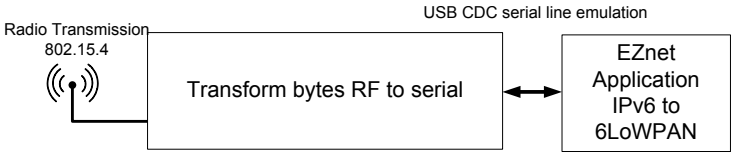


Figure 4: Schematical setup for the Raven USB stick with serial line firmware and EZnet

gateway. In the rest of the paper we explain how EZgate supports cross layering and a flexible configuration of a gateway that can be used for building service discovery, QoS support, and proxies.

For each network interface we need to store specific information related to this interface. This is the case for the data link layer, ARP layer (Address Resolution Protocol for IPv4) and ICMPv6 (Internet Control Message Protocol Version 6). Those layers for example store information about the neighborhood of each connection on a link and thus they are associated with their corresponding data link layer. If a packet is sent from EZgate TCP layer for example it passes it to a router object which determines the right network interface based on the address information, subnets or passing the packet to the default gateway. There is a single router instance in the stack which is created during stack initialization. Configuration of this stack is controlled via Java Property files which consist of key-value pairs.

We will extend our gateway for connecting wireless sensor nodes to the Internet with the Raven stick hardware presented in Section 2 as depicted in Fig. 4. It will use the same Raven stick hardware with another firmware that allows a more flexible protocol development. With this approach the stick emulates a serial interface which can be easily used from the Java RXTX library. It sends out the bytes over the air which it receives from the serial interface or handles received frames over the air to the serial interface. Finally it handles the frames from and to the listening Java process.

In this approach we will implement the 6LoWPAN protocol in EZgate.Here we use the feature to modify the stack and assemble the stack flexible suited to our needs. Thereby, we can replace the Ethernet layer in our stack with a self-made 802.15.4/6LoWPAN layer. If there is a change or an extension of the 6LoWPAN standard, the software can now easily be replaced with an improved implementation of 6LoWPAN. The stick's firmware is very generic and resource consuming buffering and protocol inspection runs on a powerful PC which improves the stability and performance of the gateway.

An overview of the different yet and future supported network interfaces is available in Fig. 5. The figure shows an Ethernet and a 6LoWPAN adapter. The 6LoWPAN adapter connects the stack to a sensor network. We will provide more detailed explanations of the Applications shown in Fig. 5 in Section 5.

5 Cross Layering with EZgate

Before we describe the cross layering support of EZgate we introduce the way networking applications use the Java Stack. The original design of EZnet allowed users to write Java applications using UDP or TCP sockets in a transparent fashion. This means that an application can be

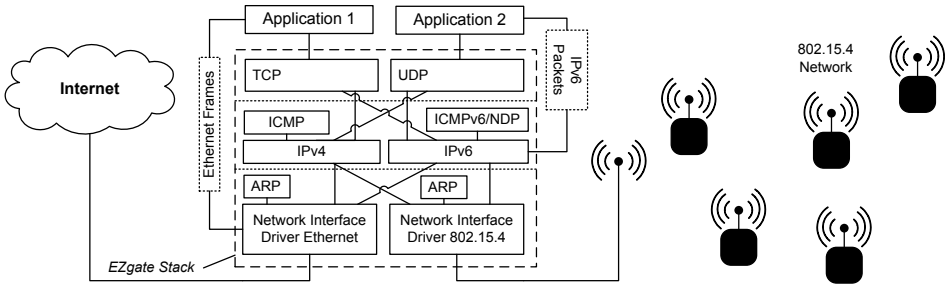


Figure 5: EZgate architecture example showing two network interfaces and two cross layer applications on top

```
public class IPCounter extends ProtocolLayer {

    private int counter;

    // Callback called if an IP packet is received
    @Override
     public void handle(Packet p) {
         counter++;
         System.out.println("Current packet count: " + counter);
     }

    public int getNumber() {
        return counter;
    }
}
```

Figure 6: Example cross layer application IPCounter counts IP packets on one network interface

compiled with a standard Java compiler and then started with EZnet. EZnet provides its own socket implementations for TCP and UDP which can be transparently used by the networking applications. Therefore, the EZnet socket interface implements the standardized network socket interface in Java.

In its original implementation a single application at a time can run with EZnet. We extended EZnet to run several applications as depicted in Fig. 5. All applications access the socket functionality as transparent as before. Fig. 5 gives an example were Application 1 has a TCP socket and also handles Ethernet frames. Application 2 has a UDP socket and also handles IPv6 packets (like our application in Section 6).

If we need cross layer functionality, the software developer writes his own handler in a class that extends the ProtocolLayer class and plugs this handler as a layer at the required position into the stack. We explain the basic steps used for creating a cross layer application with a simple IP counter example.

Fig. 6 shows the source code of an application class that counts received IP packets. The packet is passed as parameter to the method `handle()` and can be processed further in the body of this method.

Fig. 7 shows source code that instantiates this layer and adds the packet handler to the processing chain. Thats all that is needed to write and connect a handler to the stack. In the next

```
private static IPCounter ipc;

// Fetching reference to the stack
TcpIpStack stack = bootIp.getTcpIpStack();

/* Add application as an handler to the data
   link layer for fetching IP
   packets from network interface 0 */
stack.getDataLinkLayer(0).addHandler(EthernetHeader.TYPE_IP, ipc);
```

Figure 7: Adding the handler mentioned in 6

section we will implement a performance enhancing Proxy PEP with cross layer functionality in the same manner and evaluate the performance of EZgate.

6 Performance Evaluation

In this section we evaluate the EZgate approach using the UDP protocol. UDP is preferred over TCP as it has a straight, predictable behavior where lost packets are not retransmitted and sender – gateway – receiver interaction is kept at a minimum. Thereby, we can avoid effects of slightly different implemented TCP behavior that might degrade the performance due to unoptimized interaction between communication partners. For performance evaluation we chose *throughput* and *delay* as they present the most important performance metric for any networking device including gateways. First we will evaluate the throughput and the delay of our basic implementation. In the next step we demonstrate how extension to a cross layer gateway results in performance enhancements for the throughput.

The test setup for our measurements is shown in Fig. 8. The gateway process is either run natively using the OS networking stack and the Java default socket implementation (labeled ”Native Gateway”), or with EZgate on the host machine (labeled ”EZgate”), see Fig. 8(a). Sender and receiver are located on separated virtual machines on the host. This simple setup allows a quick setup of two disjoint networks connected over the gateway without setting up a real network. Since we do not observe network related matters the use of virtual machines do not affect the soundness of our our evaluation. The gateway connects to these two VMware[6] network adapters. For a reference to our measurements we also connect sender and receiver directly (see Fig. 8(b)) which we expect to deliver the best performance results (labeled ”Direct”).

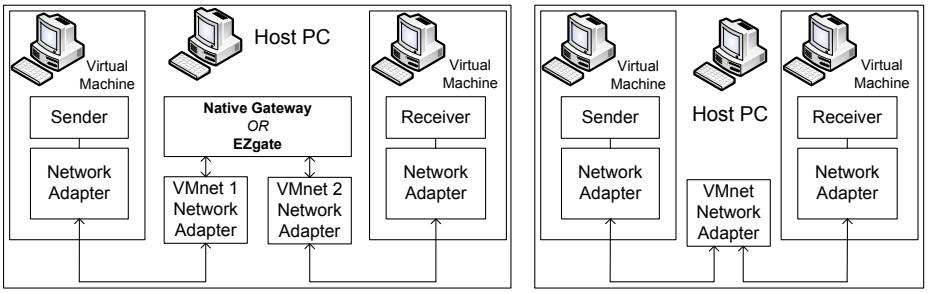
This results in three basic scenarios for evaluation.

6.1 Throughput

For the measurements we implemented a simple user space gateway which passes UDP packets received at the first network interface to a second interface and thereby connect a sending client and a receiving server to it. The sender and receiver implementations are optimized for maximum throughput.

Then we enhance our gateway with cross layer functionality by EZnet. Therefore, we implemented a packet counter for incoming UDP packets and use this information to invoke the

[6] http://www.vmware.com/



(a) Sender and receiver connected via gateway (b) Sender and receiver connected directly

Figure 8: Test setup for evaluation

`read` method of the Java UDP DatagramSocket to collect a number of already received packets, merge them into one packet and send the larger packet to the server. Normally invoking the blocking `read` is a performance risk as when no more packets are available the process will block indefinitely long until new packets arrive. However, the packet counter provides us with additional information about the number of received packets so we can safely invoke the `read` for all received packets as we know exactly how many packets are received. This enhancement is expected to increase the throughput of the system.

The results are shown in Fig. 9. The figure shows the throughput against the user datagram size sent from the sender.

Additional to the aforementioned three basic scenarios for evaluation we perform also: EZgate cross layer application collecting packets and resending packets of size 400 Bytes (labeled ”EZgate 400 Bytes merged”) and EZgate with cross layer application collecting packets and resending packets of maximum size (labeled ”EZgate maximum Bytes merged”). For example the gateway merges 14 datagrams with 100 Bytes of user data from the sender and forwards a datagram of 1400 Bytes. At 400 Bytes the gateway can merge up to 3 datagrams to a datagram of 1200 Bytes.

The maximum packet size is set to 1500 Bytes (1472 Bytes of user data for UDP) because larger packets will lead to fragmentation anyway because 1500 is the Ethernet’s MTU (maximum transmission unit). Therefore, for packets with 1500 Bytes this optimization will not have any more effect.

Fig. 9 shows that increasing user datagram size results in increasing throughput. Increasing user datagram size leads to a reduced message overhead and decreasing amount of messages which had to be processed per transmitted byte. As expected the direct connection performs best with highest throughput followed by the native gateway. The EZgate cannot reach the throughput of the native gateway. However a throughput of maximum 10 Mbps will be enough for our targeted application to build a gateway for Ad-Hoc or Sensor networks as the wireless network cannot handle this high throughput anyway. Fig. 9 illustrates how the performance improve with the cross layer feature. The datagrams collected up to 400 Bytes and merged into one datagram the throughput increases linear until a user datagram size of 200 Bytes. If the sender sends datagrams of 400 Bytes the situation is similar to EZgate (without cross layer application). With merging datagrams up to the maximum packet size the throughput is similar to the native

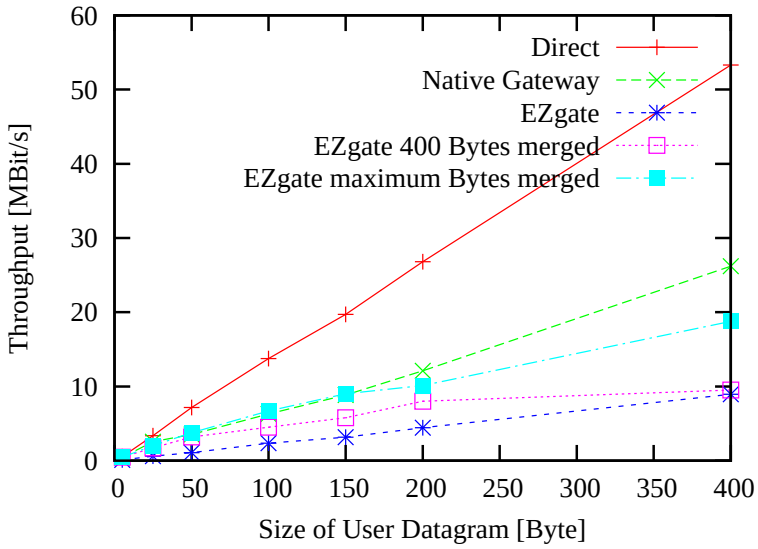


Figure 9: Gateway throughput

Method	Mean [ms]
Direct	0.37
Native Gateway	0.55
EZgate	1.04

Table 1: Delay measurements mean values

gateway.

6.2 Delay

For delay measurements the sender sends a message to the receiver. The receiver acknowledges a message by resending it back to the sender. The measured delay is the time difference between the message sending and arriving of corresponding acknowledgment at the sender. The sender sends a new message if the last is acknowledged successfully or a timeout occurs. We repeated this measurement for 10000 times for statistical soundness.

Fig. 10 shows the frequency of the delays. We experience some unexpected high delays from time to time which we identified as CPU scheduling effects as the delay was always a multiple of 16 ms. These heavy tailed delays beyond 3 ms are not plotted in the graph. The resulting mean values are depicted in Table 1. We use the same three basic scenarios than for the throughput evaluation. Also here the direct connection performs best with 0.37 ms delay on the average. The mean delay of EZgate increases approximately 0.5 ms compared the native approach. However a delay of 1 ms is insignificant compared to long distance connections from the Internet or compared to the delay in the wireless network.

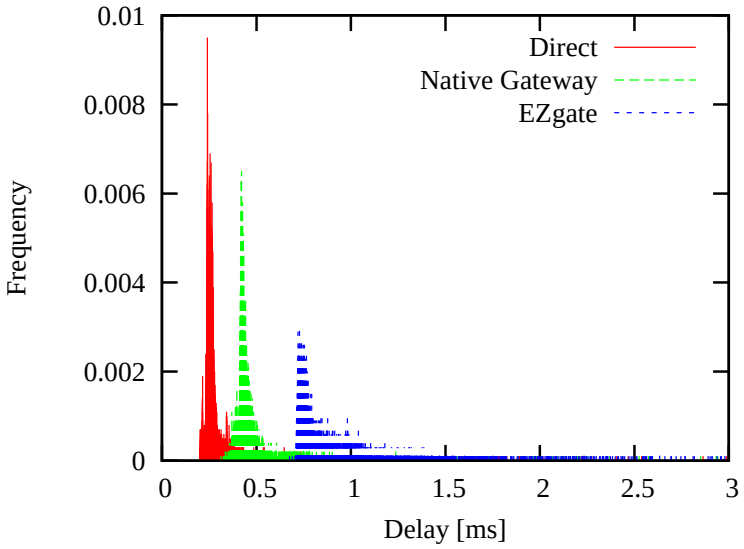


Figure 10: Gateway delay

7 Conclusion and Future Work

In this work we have presented the user space gateway EZgate based on EZnet. We discussed the extensions for a gateway in detail and described how to attach gateway applications to the stack including cross layer functionality. As we target EZgate for usage in productive environments, we evaluated the delay and throughput performance. The average delay increases by approx. 0.5 ms which can only be detected in a LAN scenario with high resolution timers. As we expect the delay within the wireless Ad-Hoc network and the Internet (due to longer distances) much larger, this slight increase can be neglected as it cannot be detected at all in a global scenario.

The maximum throughput of EZgate is less than the native gateway but with up to 10 Mbit/s acceptable for our target application where we do not expect to have more than a couple of Mbit/s of traffic load. Remember that one of the target fieds is integrating sensor networks into the Internet.

In summary EZgate is a promising approach that we will continue for our research. Future work will cover further optimizations and extensions to the networking functionality and performance of EZgate.

Planned extensions and improvements for the near future are configurable user friendly stack configuration, providing default configurations for UDP/TCP proxying, and integrating complete Java based 802.15.4/6LoWPAN support.

We are integrating EZgate in the context of the Real-World G-Lab testbed. Some features like Router advertisement, solicitation, redirection, multicast listener, error messages and support for all IP headers are not required for our current research. This functionality will be implemented on demand. In the process of this G-Lab project we start the porting of EZgate to an embedded Linux device.

Acknowledgements: This work was funded by the Federal Ministry of Education & Research of the Federal Republic of Germany (Förderkennzeichen 01BK0905, GLab).

Bibliography

[CB10] N. M. K. Chowdhury, R. Boutaba. A survey of network virtualization. *Computer Networks* 54(5):862 – 876, 2010.

[Dav08] K. David. *Technologies for the Wireless Future: Wireless World Research Forum, Volume 3*. Wiley Publishing, 2008.

[DGV04] A. Dunkels, B. Grönvall, T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*. Tampa, Florida, USA, Nov. 2004.

[EM95] A. Edwards, S. Muir. Experiences implementing a high performance TCP in user-space. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*. Pp. 196–205. ACM, New York, NY, USA, 1995.

[ESW01] D. Ely, S. Savage, D. Wetherall. Alpine: a user-level infrastructure for network protocol development. In *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems - Volume 3*. USITS'01, pp. 15–15. USENIX Association, Berkeley, CA, USA, 2001.

[GCM+08] J. F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting, A. Toncheva. The Diverse and Exploding Digital Universe. *An IDC White Paper* 2, 2008.

[JMSG07] V. Jacobson, M. Mosko, D. Smetters, J. J. Garcia-Luna-Aceves. Content-Centric Networking: Whitepaper Describing Future Assurable Global Networks. Response to DARPA RFI SN07-12, 2007.

[MKHC07] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks (RFC 4944). Sept. 2007.

[SC05] A. Smirnov, T. Chiueh. A User-Level Development Environment for In-Kernel Network Protocol/Extension Implementations. ECSL Research Seminar, 2005.

[Ste07] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), Sept. 2007.

[WF02] U. Walther, S. Fischer. EZnet: A Framework for Rapid Protocol Protyping. In *Joint Conference - ICWLHN and ICN 2002 - Networks*. Pp. 523–534. World Scientific, Atlanta, Georgia, USA, 2002.