Final Report: TCP / IP meets Fennec Fox

Sabari Kumar Murugesan, Kshitij Raj Dogra

Abstract

It has been shown that TCP/IP network architecture can be implemented on wireless embedded devices [1] [2]. The primary benefit of TCP/IP architecture is a support of networking standard familiar to industry [2] and thus promissed product vendors interoperability [3]. In this report we discuss the design, implementation, and evaluation of our TCP/IP implementation on Fennec Fox platform [4]. We are proposing an extensible system architecture for the Network stack of Fennec Fox that is modular, yet scalable for future innovations. We have implemented TCP and UDP for providing both guaranteed and best-effort services at the transport layer. Our implementation of TCP supports the essential features such as reliablity, in-order delivery and flow-control over duplex data channels. We also propose an On-Demand routing protocol, that can support multiple hop packet routing in the network. Finally we developed sample applications to demonstrate the features. Our work shows promissing results, and aims at providing network communication for future research applications.

1 Introduction

By integrating embedded systems with wireless technologies we can envision the benefits that could revolutionize our day to day life. These technologies range from more sophisticated implementations in industries like transportation, infrastructure protection to something that influences our everyday life like emergency response systems. For example, Stankovic [5], envisions smart nodes applications such as health care (Sensor Information Systems for Assisted Living (SISAL))and natural disasters (Emergency Response Systems). They would not only make industries more efficient but also help improve the quality of life. With the technology in place we could have a platform for future innovations, and applications not imagined today could be a reality.

TCP/IP are the underlying layers for communication in the modern day inernet. These layers exist below the application layer in the TCP/IP stack and are responsible for a connection-oriented, reliable connection between devices present at long distances. IP is responsible for moving packet of data from node to node in a network and TCP to ensure correct delivery. In the stack they are referred to as the Network layer and Transport layer respectively. As the TCP/IP suite these protocols provide support for complex applications to be built and shared across the web.

The developments in wireless sensor networks allow IP networks to operate over small-embedded devices. First Duckels has shown how to implement full TCP/IP stack on 8-bit architecture [1]. More recently, Hui and Culler proposed to use IPv6 address architecture in low power wireless devices [2]. TCP/IP has been standardized in [6].

For our research we have used Fennec Fox as the platform. Fennec Fox is a platform providing network reconfiguration mechanism [4]. The platform consists of layered network protocol stack, where at runtime various library module may support each layer service. As part of our work, we have restructured the Network layer of the stack and implemented library modules to provide transport layer and IP layer services. Prior to this effort, the network layer was monolithic and offered basic routing mechanisms. In order to implement transport layer protocols such as TCP and UDP in a reconfigurable manner, the existing Network layer is divided into the following three layers: IPNet, Transport and Routing. IPNet is an encapsulating layer which provides network services to upper layers, bridges the communication between Transport and Routing layers and utilizes the link layer services. This design was modular and offered configuring the network layer in various flavors. Eg. The Network layer

1. could be running UDP over IP implementing fixed path routing algorithm or
2. could be running UDP over IP implementing On Demand routing algorithm or
3. could be running UDP over TCP implementing fixed path routing algorithm or
4. could be running TCP over IP implementing On Demand routing protocol.

The library modules for TCP/IP stack was then developed over this new network stack to offer a robust, reliable and scalable service for developing industry standard applications for sensor networks. We have also developed few applications that emphasizes the need for a dynamic transport service which serve the dual purpose of being a model for creating industry oriented applications and for regression testing the platform. While developing these modules, we were aware of future requirements that may warrant dynamic addressing schemes and variable length packets.

In this document, we would first look into the current architecture of Fennec Fox and then explore the alternative designs that were considered but later withdrawn owing to various constraints while implementing them. Then we would describe the proposed architecture for the Network layer offering transport and routing layer functionalities. The libary modules for TCP/IP stack would be explained in length focussing more on its diverging aspects from a conventional implementation. Finally, we state the future course of actions that, if implemented, would greatly enhance the capability of the Fennec Fox platform for developing industry grade systems.

2 Fennec Fox: Current Network Stack

Fennec Fox provides a network communication protocol stack with four layers [7], as shown on Figure 1. Each layers is supported by a set of alternative modules, each implementing the full layer functionality while being optimized for a target performance metric. At runtime, Fennec Fox dynamically adapts the network to the given changes in the application scenarios by reconfiguring the combination of modules that are executed for each layer.

In order to be compatible with the Fennec Fox network protocol stack, a library module must provide and use the interfaces of the layer that it becomes part of. Specifically, a module provides an interface to the upper layer by implementing the functions whose definitions are part of that interface. A module, instead, uses the interface provided by a lower layer by calling functions defined by the interface and by implementing signal handlers that are sent at runtime from the lower-level modules. Figure 1 shows the top-down bottom-up message exchange interfaces between the layers of the network stack.

The figure also depicts horizontal communication between the modules. Both network and MAC type library modules are supported with the same or two different addressing library modules, because Fennec Fox decouples communication problems from addressing. The Fennec Fox Control Unit uses management interface Mgmt to start and stop execution of modules across the layers of the network stack. The Control Unit is also responsible for executing network policies, keeping track of events, and synchronizing all network nodes to the same configuration, using Trickle-based state synchronization protocol.

3 Fennec Fox Architecture

3.1 Proposed Architecture

We suggest the following TCP/IP stack (Figure 2) for Low Power Wireless Devices.

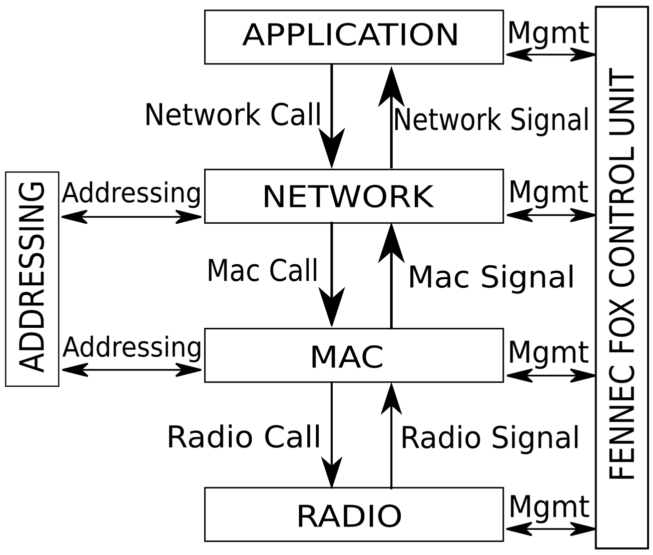


Figure 1: Current Fennec Fox Architecture

This implementation fared well as we were able to define TCP, UDP and Routing as separate components. Each component is supported by a set of Interfaces which defines its functionalities. They are individually attached to a pseudo Mac layer, which is being provided by the Network layer (IPNet). This fact makes individual components act as independent layers. Hence we provide flexibility to our architecture as we leave scope for further additions of protocols to our stack. For example we may like to introduce network protocols like DYMO, AODV etc. which can easily co-exist in this architecture (as an additional module). The details and wiring of the component wiring can be viewed from the Appendix.

1. Application - This is the top layer of the stack. All programs are implemented here and use the NetworkCall and NetworkSignal interfaces to communicate with the layers below.

2. Network (IPNet) - This is the network layer and relays interface calls to TCP, UDP or Routing. Following two interfaces are defined for this layer and are used by the application layer

   - **NetworkCall** -
     command uint8_t getPayload(msg_t *msg)* - Gets the payload of the message.
     *command uint8_t getMaxSize(msg_t *msg)* - Sends the message to the MAC.
     *command error_t send(msg_t *msg)* - Gets the maximum size of the packet.
   - **NetworkSignal** -
     *event void receive(msg_t *msg, uint8_t *payload, uint8_t len)* - Signal is triggered when

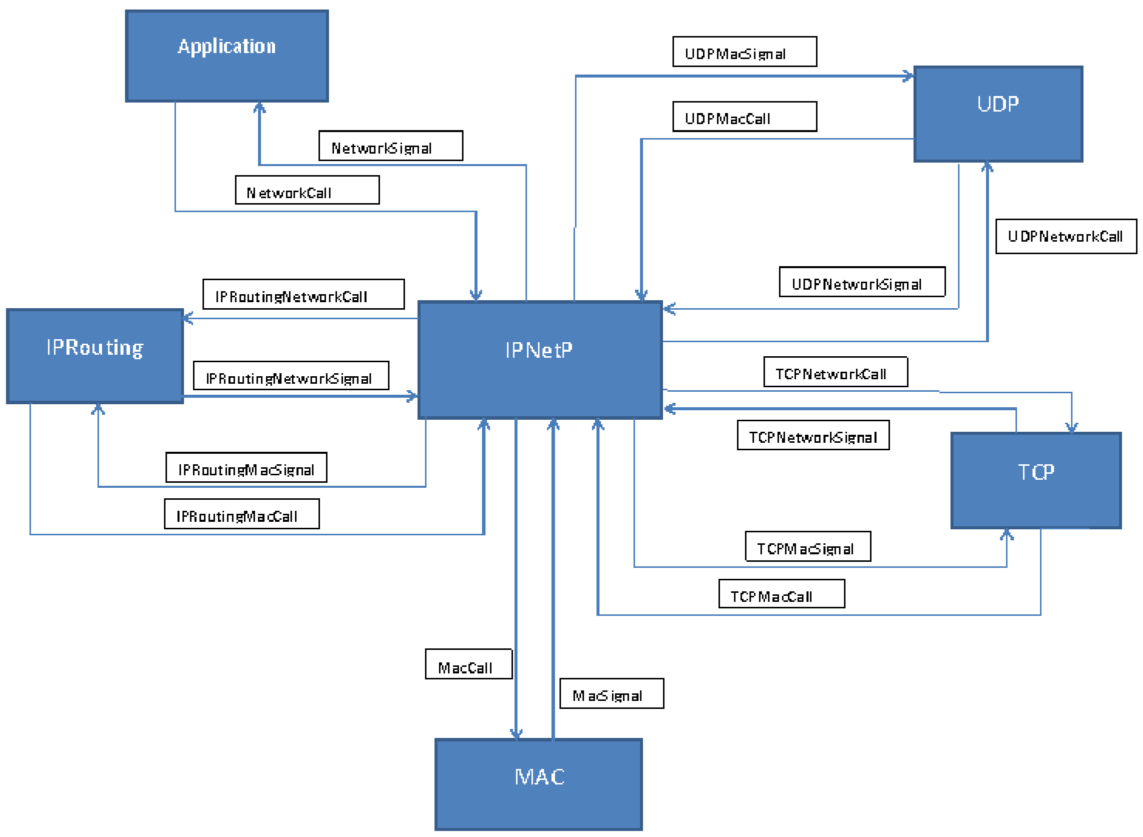


Figure 2: Current LPWN

Network receives data packet.
*event void sendDone(msg_t *msg, error_t err)* - Signal is triggered when Network sends a data packet.

This layer is wired to MAC and is thus responsible for transmitting data. netw

3. Transport It consists of two modules TCP and UDP. We have defined two interfaces which are provided by both the modules. The same interface is used by IPNet.

   - **TransportCall** -
     command uint8_t getPayload(msg_t *msg)* - Gets the payload of the message.
     *command error_t send(msg_t *msg)* - Sends the message to the MAC.
     *command uint8_t getMaxSize(msg_t *msg)* - Gets the maximum size of the packet.

   - **TransportSignal** -
     *event void receive(msg_t *msg, uint8_t *payload, uint8_t len)* - Signal is triggered when the transport module receives a data packet.
     *event void sendDone(msg_t *msg, error_t err)* - Signal is triggered when the transport module sends a data packet.

   It was a design decision that was taken to include new interfaces for the transport layer. This addition gives us the flexibility to add more functions like
   *command uint8_t *getSourcePort(msg_t *msg)*
   *command uint8_t *getDestinationPort(msg_t *msg)*
   We discuss more on this in the Future Scope section.

3.2 Earlier Architecture Proposal

We suggest the following TCP/IP stack for Low Power Wireless Devices (Figure 3).

This was a simplistic design that was engineered. It followed from the conventional TCP/IP stack as all modules were acting as a Transport layer and IPRouting layer. It was good for understanding purpose but had a major design flaw. Owing to the limitation of FennecFox programming, we were unable to direct the IPRoutingSignal to either TCP or UDP. Also the design had limitations if we would scale our application as discussed earlier.

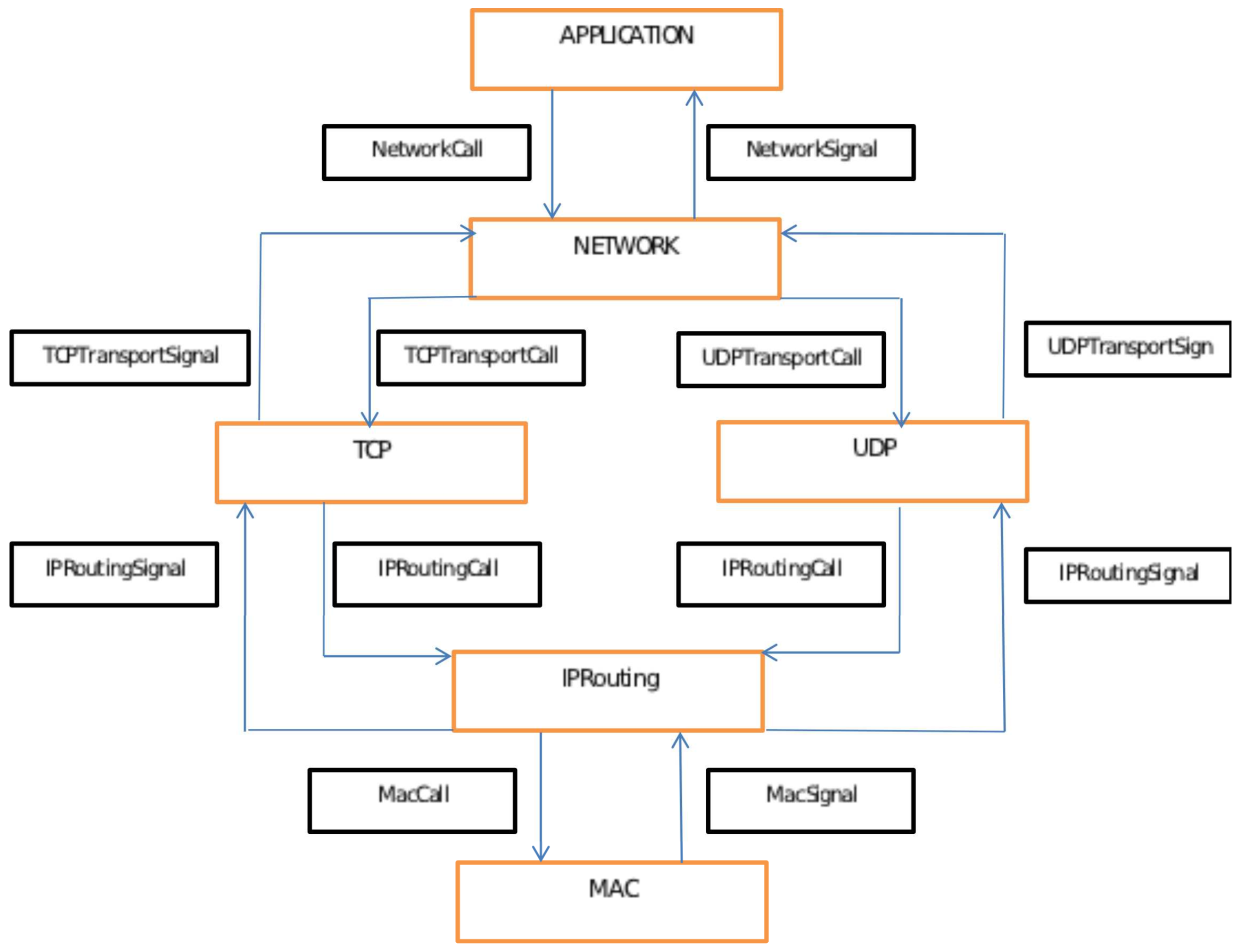


Figure 3: Old Architecture

4 TCP/IP Design on Fennec Fox

4.1 Motivation

Internet Protocol (IP) is the most important protocol for networking and it's ubiquitous reach in the world of Internet is the driving factor for the development of a similar stack for Low Powered Wireless Networks (LPWN). We cautiously use the word similar because of the design choices that we have made in implementing the TCP/IP stack for wireless networks. Our implementation is neither nave nor is full-fledged in it's approach. We have not followed certain conventions and traditions but we have kept in mind the services that TCP/IP offers and have implemented them in a different manner.

LPWN and its applications are quite different from those of local and wide area networks. Such networks have traditionally been used for area monitoring, environmental sensing, industrial control etc. The difference is usually in the amount of data that is generated and the transmission delay that is tolerated. Often such systems require effective delivery of data in a reliable and timely manner. The size of the data generated by such low powered devices may be lesser than the size of the overheads involved in the TCP/IP protocols. Using a traditional protocol's header and implementing all of its features may prove costly in this scenario. We have implemented TCP/IP by maintaing the important aspects of the protocols but at the same time considering the constraints in a LPWN system.

We shall now look into detail on how we provide the support for Transport layer on Fennec Fox by explaining in detail the implementation of two important protocols TCP & UDP. Transmission Control Protocol & User Datagram protocols provide connection-oriented and connection less transport services to the higher layers, respectively. We will elaborate the fundamentals and the design changes that we have made in both cases.

4.2 TCP on Fennec Fox: Fundamentals

TCP is a fundamental protocol that provides the core features in the TCP/IP stack. The following are the important characteristics of this protocol offered on Fennec Fox:

Duplex: Bi-Directional data transfer

Like the traditional implementation, both the connected devices can transmit data simultaneously over the same TCP connection. Our system has been optimized for simplex and duplex communication. Please read the concept of instant acknowledgements.

Reliable - Guaranteed Delivery

If TCP accepts data from the application, it is guaranteed to deliver it to the receiver. It acknowledges every delivery and maintains a cache to resend the data upon failure. The retransmission mechanisms will be explained in detail in the following sections. This feature is fundamental to TCP's implementation and is retained as it is.

In-Sequence Delivery

Not only does it ensure reliable delivery, it ensures that the data is delivered in-order to the concerned application. The receiver maintains a cache for out-of-order deliveries, sequences the packets and forwards it to the application. This is a huge advantage of TCP over the connection-less protocols like UDP.

Eliminates Duplicate Deliveries

TCP makes sure that the recipient receives the data only once from a sender by dropping duplicate segments. Duplicate packets can be worse in mission critical applications.

Flow Controlled - Managed data transfer

Our implementation of TCP is also flow controlled, that is, it verifies whether the recipient is capable of accepting the data at the rate at which it is being sent. If the sender is experiencing delays in the acknowledgement of data from the receiver, then it adjusts its sending rate to match the processing rate of the recipient device.

Congestion Control - To be Implemented

4.3 TCP Operations

4.3.1 Data Handling & Packaging

Moving away from the conventional stream-oriented TCP, we here deal with datagrams of fixed sizes which are determined at the application level. We accept a message in the form of a datagram from the Application, add TCP header and pass it down the layers. LPWN consists of energy constrained devices that would not usually pump-in a continuous steam of data for long durations. As a result, waiting for a minimum transferable segment of data from the application would cause inherent delay in sending the information. Also, this may increase the number of transmissions required. For Eg. Consider a temperature sensor relaying the temperature reading to a central controller every few minutes. We would not want the sending device to wait until the application collects a certain amount of information as the data we deal is delay sensitive. Thus, the overhead of packaging is moved to the application itself. The application can determine the size of the transferable unit that it wishes to send at a given time. This choice of design has two-fold advantage in low powered wireless networks:

1. Constant data to header ratio: By fixing the segment size we maintain a fixed data to header ratio for each segment. This ensures that we do not experience issues such as unnecessary transmission overheads when sending small amount of data and minimizes the number of transmissions required when data arrive at unknown sizes. Though there are ways to avoid this scenario, they often come at the cost of delays in waiting for the next byte stream which is explained next.
2. Avoids Segmentation Delay: Out approach ensures that we do not experience segmentation delays due to dynamically changing application data rate. Eg. Consider the case of an application that is sending 152 bytes of data over an interval. As the application writes data in the TCP buffer, TCP segments it into chunks of 50 bytes each. After the third segment, TCP now only has 2 bytes of data to be sent. Thus, TCP waits for a while before sending this information, hoping that the application would push some more data. At the other end, the receiver may not be able to make sense of the information that the sender has sent so far without this missing piece. We believe that in an LPWN environment, the application would be more intelligent in the packaging the data that it wishes to send rather than the layer that handles data transfer.

4.3.2 Data Processing: Identification

The process of Data identification is using the concept of Sequence numbers. Every transferable unit is assigned a unique sequence number by the sender which is used as a reference for all future correspondence. Sequence numbering helps us to achieve the following:

1. Reordering of packets at the receiver
2. Sending & Interpreting Acknowledgements
3. Maintaining sender & receiver window sizes (Flow Control)
4. Avoiding Duplicate Packets

4.3.3 Data Transport: PAR & Sliding Window

1. **Reliability through PAR:**

   Positive Acknowledgement with Retransmission (PAR) is a mechanism TCP uses to provide a reliable channel for data communication between devices. This is a closed loop mechanism, in that, the receiver keeps updating the sender about the status of the data received. This feedback is taken into account by the sender to detect data losses and retransmit the information. Eg. The sender sends a TCP segment to another device and stores the data for future retransmissions. Upon receiving the data, the receiver sends an acknowledgement to the sender. The sender sees this acknowledgement and purges the saved data. If the packet is lost or the acknowledgement fails to reach, the sender retransmits the same data again to the receiver, hoping that it would reach the destination this time. The wait time before a packet is retransmitted is configured internally. Having said this, it would be rather inefficient to wait for ACK's for every segment of data before sending the next. Thus, we transmit the data until we hit a threshold (later defined as window). We then use the sequence number to identify each individual segment and it's acknowledgement. The same sequence number is used to reorder the packets at the receiver. Technical details on how this is achieved using buffers & timers forms the basis of the next section.

2. **Sliding Window Acknowledgement System:**

   The idea of PAR is implemented using Sliding Window Acknowledgment System and is slightly different owing to our initial design choice of using datagrams instead of data streams. Thus, instead of maintaining a window of bytes, we maintain a window of segments having fixed bytes. Sliding window can be further explained using the following classification of the transmission queue. At any given point of time, we would have a -

   A set of segments waiting to be sent

   A set of segments that have been sent but Unacknowledged

   We are not considering the following to be part of the transmission scope, but we are quoting them for the sake of completeness. They are -

   Segments that have been sent, acknowledged and dropped from transmission queue

   Segments with the application, waiting to be sent as the queue is full

   Usable Window: The set of segments waiting to be sent constitute this window definition. *Sent Window* The set of segments that have been sent but awaiting acknowledgements. *Send Window*: Usable window + the Sent Window.

3. Sliding the Usable Window
   The size of a Send Window is usually fixed to a constant value. This is the maximum number of segments a sender can process. To begin with the usable window is fixed to a constant value, which means the sender can start transmitting the data as and when it is added into the transmission queue. As and when the sender sends a new segment the window accommodates the segment which is enqueued by the application. As the usable window moves, the size of Sent window increases at the expense of the usable window. The sliding happens until the size of /emphUsable Window reaches zero. In other words, the sent window equals the maximum size the sender can process. We no longer accept or process any application data after we have reached this state. For sender to process more data, the size of Sent window should reduce.

4. Processing Acknowledgements
   The way by which the Usable window grows back, is by processing the acknowledgements sent by the receiver. As and when an acknowledgement arrives and is matched against a segment in the sent window, it is dropped from the transmission queue. This process shrinks the sent window and increases the usable window. The matching is done by comparing the sequence number of the acknowledgement with that in transmission queue. This is the only way the size of the Usable window can increase. Note that it can increase only up to the value with which it started initially.

5. Retransmission of Segments
   Every segment that is part of the Sent Window buffer has an expiration time. After which it will be moved into the Usable window buffer for retransmission. Without this feature, a segment could be indefinitely held in the transmission queue thus blocking the window dynamics.

6. Concept of Instant Acknowledgements
   Currently, our implementation does not support the concept of delayed acknowledgements. In normal TCP implementations, the receiver sends an acknowledgement in hopes of piggybacking the response data from the receiving application. But in case the application does not generate an immediate response, which is the case for most simplex communication scenarios, it waits for a short time before sending an ACK segment with no data.This delay is introduced so that a new segment is not created just for the sake of sending the acknowledgement.In our implementation, we piggyback the application data only if it is available. In other words, we send an ACK segment without any further delays. This is a design trade-off assuming most communication in wireless

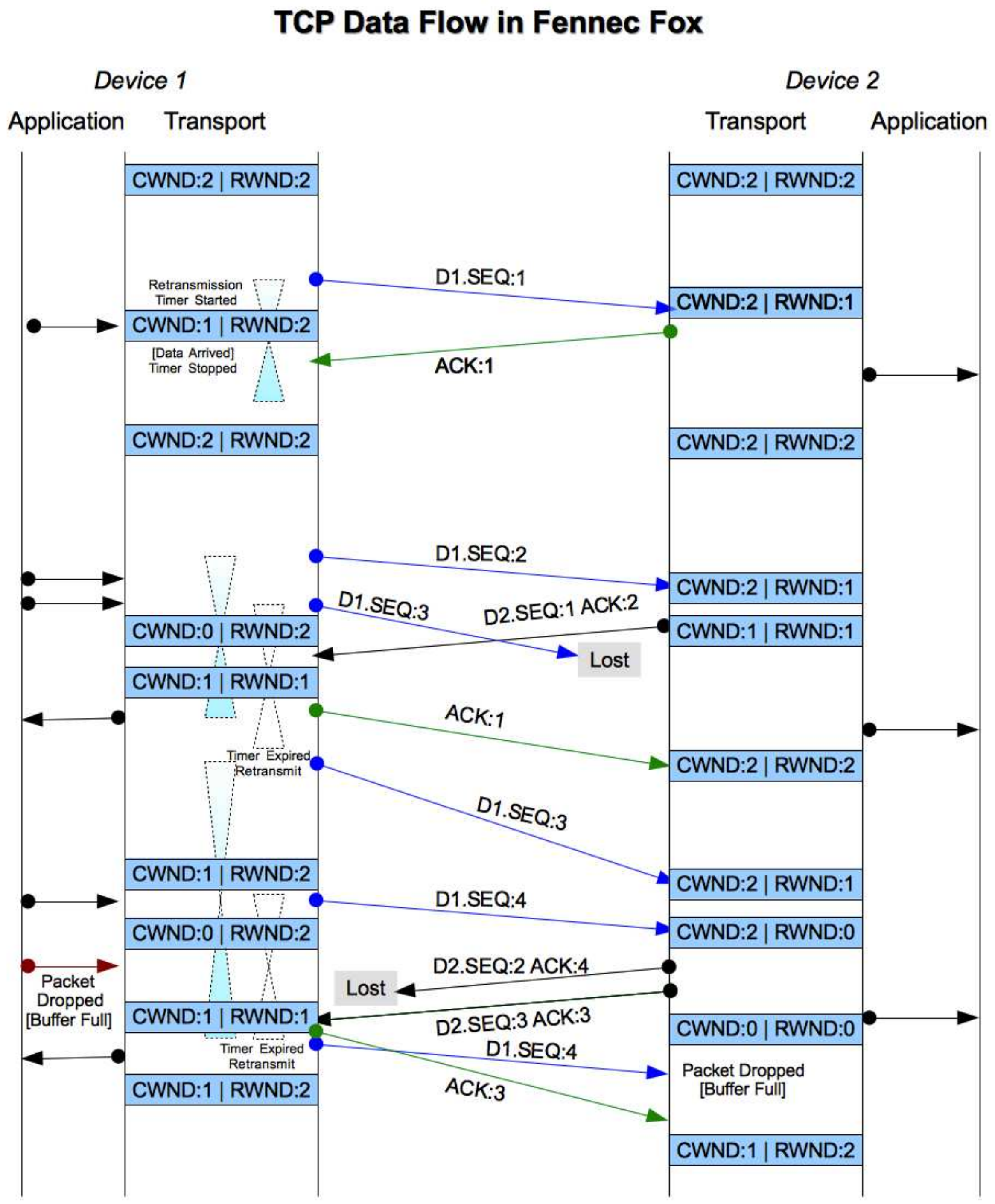


Figure 4: TCP Flow Sequence. CWND refers the size of Congestion Window (Send Buffer) and RWND refers the size of Receiver Window (Receive Buffer). The examples shows how the the window lenght affects the flow of data. When CWND is 0, the application can no longer write data to TCP's buffer. When RWND is 0, any incoming packet is dropped at the receiver. You can also notice how ACK piggybacks with DATA packets, shown by black arrows. Blue arrows represent DATA packets without ACK. Green Arrows represent ACK Packets with no DATA.

sensor networks may predominately involve simplex or half-duplex communication. We are still debating on this choice and may change the implementation in the near future.

4.4 UDP on Fennec Fox

We also implemented UDP on the Fennec Fox by encapsulating Application data into Datagrams that are stored in a buffer similar to TCP design. The only assurance that UDP provides is that the data packet is sure to exit the device, but whether it reaches the destination or not is beyond its scope. Also, at the receiving end we employ a buffer to read data that comes at a rate faster than the application can process. Also, as mentioned earlier, we do not guarantee the packets will be stored in sequential order. This effort marks the completion of the transport layer functionalities currently offered by Fennec Fox.

5 Evaluation

Our model has been tested for the following communication models built as Applications on Fennec Fox environment -

5.1 Simplex Flow - SendCounter

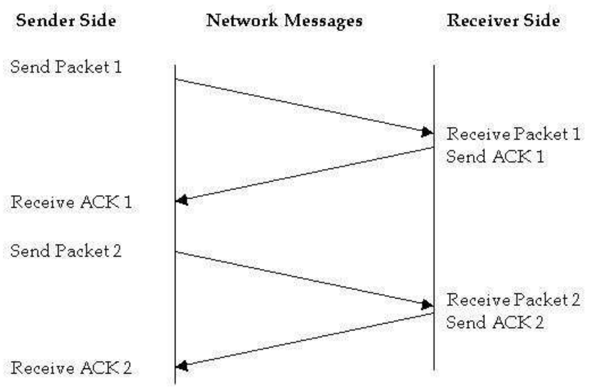


Figure 5: Simplex Data Flow. The communication is simplex at the Application layer. But TCP still sends ACK packets to acknowledge the delivery.

In the sample application, the Client device sends a counter value in incremental fashion. The server only responds with an acknowledgement to the sender at the Transport layer. In this implementation our architecture scaled well and delivered data in a reliable and sequential order. This application mainly tests the in-order delivery feature of TCP. The delay experienced in the case of simplex is failry a constant as there is no overflow of buffers and the receiver sends an immediate ACK as shown in Fig 6.

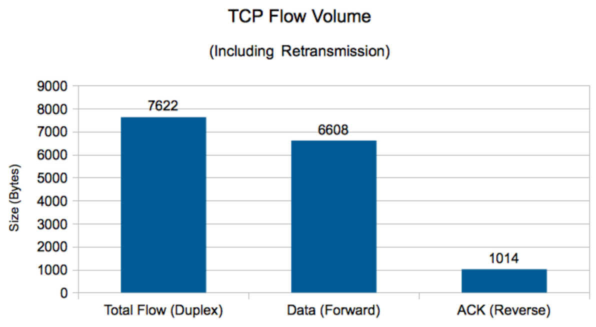


Figure 6: TCP Flow Volume : Client - Server Simplex Application. It shows that the ACK packets do occupy a fraction of the total bandwidth.

5.2 Streaming data - SendCharacterStream

This application is based on Client-Server model which tests the reliability, scalability and ordernes of TCP between communicating devices on an one-way connection. The Client node sends the Server a data stream in chunks of packets that can fit into a predefined size. As we had already mentioned, packaging of data is done at the Application layer in Fennec Fox. Here, we use utility functions which are exposed by the Underlying layers to determine the maximum size of the Application payload per packet. The data steam is reconstructed at the receiver and verified for correctness. We tested TCP for a large string of data (40KB). The data reached the server without any noise in a reliable and sequential order. This application can be mainly used for regression testing of the reliability of TCP to transport data in Fennec Fox.

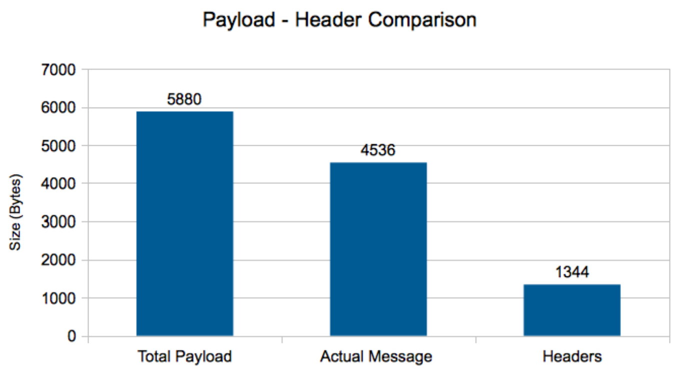


Figure 7: Payload - Header Size Comparison for SendCharacterStream Application.

5.3 Duplex Flow - PingPong Counter

This application tests the bi-directional data flow offered by TCP in Fennec Fox. This is also based on Client-Server model where in a Client sends a burst of packets with incremented counter values. The emphasis here is on an application that is expecting a fast duplex connection between two nodes. The Server receives the counter and adds a constant value to it, which can be termed as an answer to the question sent by the client device. For eg. the client device would send segments with counters of values ranging from 1 to 500, while the server has to reply with values 501 to 1000. This makes sure that two devices are able to establish a bi-directional flow over a TCP connection. This is a comprehensive test application for TCP. Fig. 6 represents a study of the latency between packets. The forward channel exhibits a wavvy pattern because the send buffer becomes full while waiting for the ACK's from the receiver. It then graudally climbs upon the receipt of the ACK's.

The following graph illustrates the latecy experienced in TCP for a stream of size 4KB.

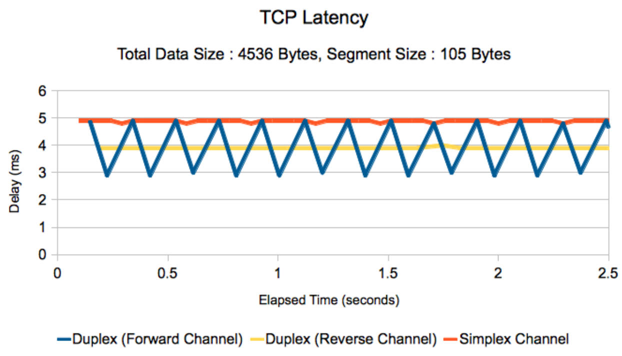
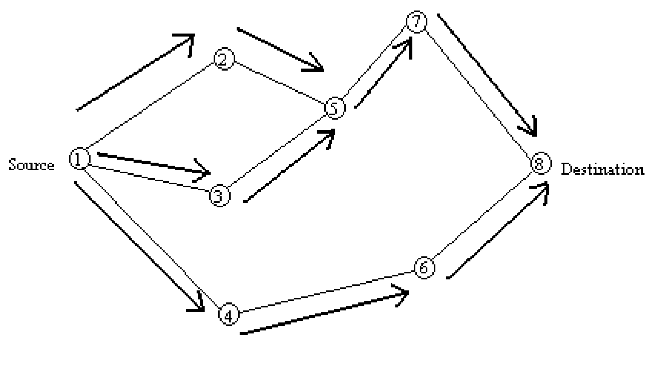


Figure 8: Latency in TCP for data stream. Delays experienced in the Streaming data & PingPong Counter applications.

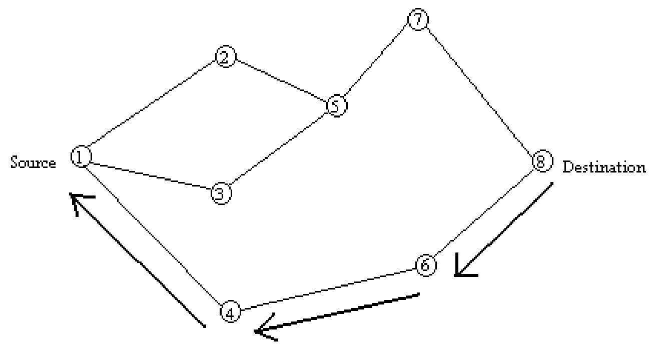
6 On Demand (Reactive) Routing Protocol

This type of protocols finds a route on demand by flooding the network with Route Request packets. In contrast, the most common routing protocols of the Internet are proactive, meaning they find routing paths independently of the usage of the paths. Figure 9 shows an example of such a protocol [8]

We have implemented the former as IPRouting. Before we explain its working, let us define two types of packets that might exist in the network at a time.



(a) Propogation of Route Request (RREQ) Packet



(b) Path taken by the Route Reply (RREP) Packet

Figure 9: On-Demand Routing Protocol

- **Data Packets** - These are the packets containing either the data sent from the application via TCP/UDP or the acknowledgements sent from the receiver.
- **Route Request (RREQ)** - This packet contains source node address (TOS_NODE_ID), destination address, hop count and the sequence number. They are used to initiate route discovery if no route already exists from source to destination.
- **Route Reply (RREP)** - Similar to the Route Request, this packet sends an acknowledgement to the source, signifying route discovery.

Apart from these packets a reverse route entry table is maintained which shall contain the source address, destination address, hop count to the source and the next hop to the source.

IPRouting is silent till the time a route is requested. When TCP sends a packet for a destination node, IPRouting checks whether a route is available for that destination. If it does not have a route entry for the destination, it broadcasts RREQ throughout the network after setting all corresponding header values. All the nodes that receive the RREQ make two observations.

Firstly, the intermediate node checks whether it has already received the RREQ by checking the sequence number. If it has already processed the RREQ then the current request is discarded. Secondly, it checks the header to determine whether it is the destination. If not then the node rebroadasts the packet after incrementing the hop count by 1 and making suitable entries in the reverse route enrty. If the receiver happens to be the destination, then it makes a Route Reply packet. This packet is unicasted to the previous node from which it recieved the RREQ by referring the Reverse Route entry table.

The advantage of using a reactive routing protocol is the reduction of network traffic. Whenever a packet needs to be sent, route discovery is performed if a path does not already exist. This reduction is substantial as the major portion of network traffic goes into link maintainance. Also given the fact that wilress smart nodes are challenged with computation power and battery life, opting for such a protocol was a necessity. Apart from the positives, our implementation has a couple of limitaions -

1. We consider the sensor nodes to be stationary. This implies that once the routes have been established there is no need to note how old the discovery is. Hence the route may never get stale and refreshed. As a result we do not use Time To Live (TTL) for each corresponding route entry. Our assumption of stationary nodes may not entirely be true.

2. Wireless nodes maintain entries corresonding to just one destination at a time. If the application changes the destination, then new routing details will be created and the older ones will be lost. This may cause a significant effect in throughput.

3. This routing protocol, measures hop count as the unit of distance between two nodes. This methodology does not corroborate the theory of signal fading as the physical distance between nodes increase.

For wiring details, please refer to the appendix.

7 Future Scope

Having implemented TCP and Routing protocol for Low Power Wireless networks, there are a number or improvements and advancements which seem important. We categorize a few proposals here [5]-

1. **Interfaces** - We plan to increase the functions currently implemented in the interfaces, so as to aid programming complex applications.

2. **Programming abstractions** - Raising the abstraction for the programmers to the point where they can directly integrate their 'C' Programs for wireless sensor devices would be a key to widespread wireless sensor network usage. APIs will be built which would take place of say, BSD sockets with respect to the Fennec Fox architecture.

3. **Addressing** - Implementing IPv4 and IPv6 addressing for each smart node.

4. **Wide Scale Implementation** When Sensor Netowkrs deal with real-world situations, data services will meet with unpredictability. Implementing TCP and Routing would provide a more reliable, robust and timely delivery of data and events along with widespread usage over several smart nodes. We now give some sample applications where sensor networks can be effectively used -

   - **Sensor Information Systems for Assisted Living (SISAL)** - Low Power Wireless Networks can assist in secure and comfortable living. Older generation lives alone and this could be a cause of health concern. SISAL uses smart devices, sensor networks and information systems which could be designed to monitor their health and report data to a medical center. Emergency services could be easily provided. This would help elderly to live safe and independently.

   - **Emergency Response Systems** - The smart nodes can be effectively used in situations such as earthquakes, surveillance, floods or terrorist attacks. These devices can be dropped from an aircraft in huge numbers and thus should be cheap. Based upon the information relayed we can take tactical decisions.

8 Conclusion

We have redesigned the networking stack of Fennec Fox and implemented TCP/IP support libaries along with a Routing protocol and few sample applications. TCP was re-engineered to suite the limitations of our environment, but none of the core features were compromised to do so. Our routing algorithm is modeled on the lines of On Demand Reactive Routing protocol family and can easily be extended and scaled to similar established protocols such as AODV (Ad-Hoc on Demand Distance Vector) and DYMO (Dynamic MANET on Demand). We have demonstrated the feasibility of such protocols on low-powered wireless networks. The results were promising and encouraging to develop novel applications utilizing the new network layer functionalities.

This project demanded a lot of research in the areas concerning low-powered wireless networks, mostly in terms of their capabilities and limitations. We had to study many existing internet protocols and scale them down for implementation in our environment. The Fennec Fox, Swift Fox & Cape Fox enviroments were new to us and we had to invest a great deal in studying their fundamental architectures and coming up design changes. We were able to grasp the programming language 'nesC' in a short span and became proficient enough to implement the libraries for Fennec Fox's networking stack. On a whole, we had a great learning experience while contributing to this project.

9 Appendix

The following shows the wiring implemented to created the TCP/IP framework.

Network layer (IPNet) relays interface calls to TCP, UDP or Routing (IPRouting).
The NetworkCall and NetworkSignal are wired individually to each of the three components.
IPNetP.TCPNetworkCall -> TCP_C
IPNetP.TCPNetworkSignal -> TCP_C
IPNetP.UDPNetworkCall -> UDP_C
IPNetP.UDPNetworkSignal -> UDP_C
IPNetP.IPRoutingNetworkCall -> IPRouting_C
IPNetP.IPRoutingNetworkSignal -> IPRouting_C

All three components are connected to the MAC, which is provided by the Network layer.
TCP_C.TCPMacCall -> IPNetP.TCPMacCall
TCP_C.TCPMacSignal -> IPNetP.TCPMacSignal
UDP_C.MacCall -> IPNetP.UDPMacCall
UDP_C.MacSignal -> IPNetP.UDPMacSignal
IPRouting_C.IPRoutingMacCall ->
IPNetP.IPRoutingMacCall
IPRouting_C.IPRoutingMacSignal ->
IPNetP.IPRoutingMacSignal

References

[1] A. Dunkels, "Full TCP/IP for 8-bit architectures," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, ser. MobiSys '03. ACM, 2003, pp. 85–98.

[2] J. Hui and D. Culler, "Ipv6 in low-power wireless networks," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1865 –1878, November 2010.

[3] J. K. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler, "Beyond interoperability - pushing the performance of sensor network ip stacks," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '11. ACM, 2011.

[4] M. Szczodrak and L. Carloni, "A complete framework for programming event-driven, self-reconfigurable low power wireless networks," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '11. ACM, 2011, pp. 415–416.

[5] J. Stankovic, I. Lee, A. Mok, and R. Rajkumar, "Opportunities and obligations for physical computing systems," *Computer*, vol. 38, no. 11, pp. 23–31, November 2005.

[6] R. Braden, "Requirements for internet hosts - communication layers," 1989.

[7] M. Szczodrak and L. Carloni, "Embedded system level design group," http://embedded.cs.columbia.edu.

[8] P. Misra, "Routing protocols for ad hoc mobile wireless networks," http://www.cse.wustl.edu/.