Micrium OS Network

Agenda

- Micrium Overview
- Micrium OS Network Introduction
- OSI Model Overview
- UDP Lab
- Network Applications Overview
- HTTP Server Lab Overview

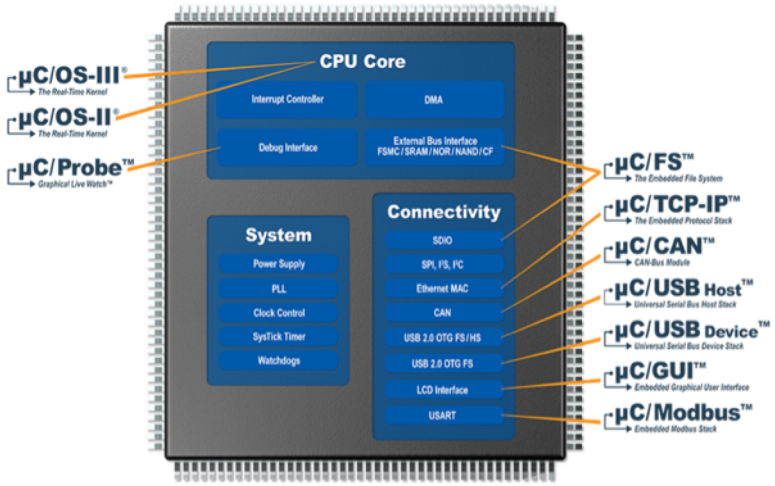
Micrium Overview

Background

- Micrium was founded in 1999, acquired by Silicon Labs in 2016
- Main office in South Florida, with an additional facility in Montreal
- Provider of high-quality embedded software
  - Known for remarkably clean code, thorough documentation, and top-notch technical support
  - A full lineup of products, including real-time kernels, protocol stacks, a file system module, and GUI software

Who Are We?

- An Embedded Software Company
  - Proven - Shipping uC/OS For Over 25 Years
  - Most Widely Deployed RTOS (UBM 2015 Survey)
  - Strong in Safety Critical Applications
  - High Quality and Robust Code



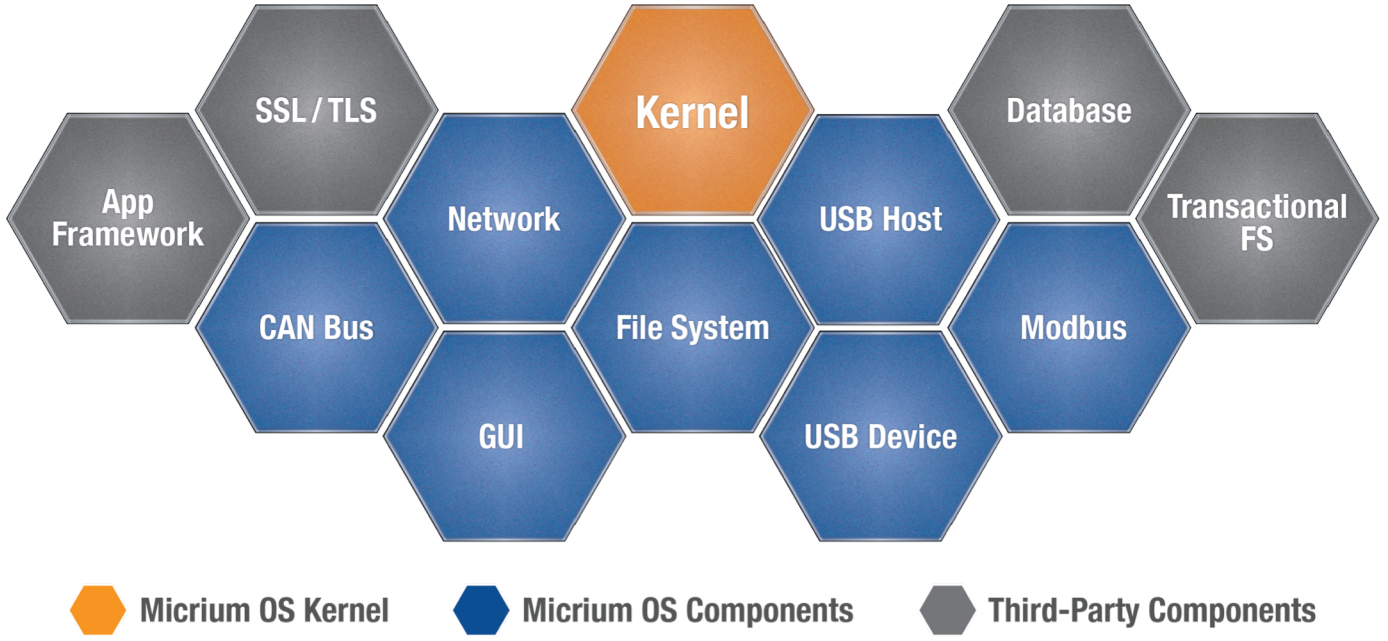
RTOS

Debug Tools

Education

Micrium OS

- A full-featured embedded operating system
- Released in 2017
  - EFR32 and EFM32
- Not a single package intended for use on all Silicon Labs devices
  - Few dependencies among the components allows developers select the software modules that make sense for their project



Micrium OS Kernel

- Based off the **extremely** successful µC/OS-III kernel
  - Updated error handling
- A **reliable** kernel with an efficient, preemptive scheduler
  - Supports round-robin scheduling at each task priority level
- Supports an **unlimited** number of tasks and other kernel objects
- Highly configurable
  - Gives the ability to enable and disable most parts of the kernel to save space
  - ROM size ranges from **6-24 kBytes**
  - RAM size is typically **3-4 kBytes**
- Delivered in source-code form, and its thorough documentation helps to ensure a smooth user experience
- Built-in performance measurement capabilities

Micrium OS: Communication Software

- **Network**
  - Features dual IPv4 and IPv6 support, an SSL/TLS socket option
  - Supports DHCP, DNS, HTTP, MQTT, SNTP, Telnet, SMTP, TFTP, FTP
- **USB Device**
  - Support for Audio, CDCACM, CDCEEM, HID, MSC, and Vendor classes
- **USB Host**
  - A USB host stack for embedded systems equipped with a USB host or OTG controller.
  - Includes support for MSC, HID, CDC ACM, USB2Ser and AOAP classes
- **CANopen**
  - High-level communications protocol and device-profile specification based on the CAN protocol
  - Aimed at embedded networking applications, such as in-vehicle networks
  - Widespread protocol used in various industries, especially in automation and motion applications

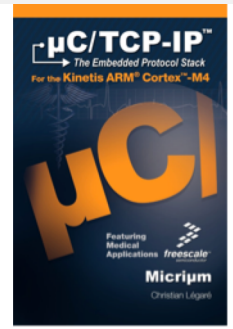
Micrium OS: Storage Software

- **File System**
  - A FAT file system compatible with a wide range of storage devices. An optional journaling component provides fail-safe operation

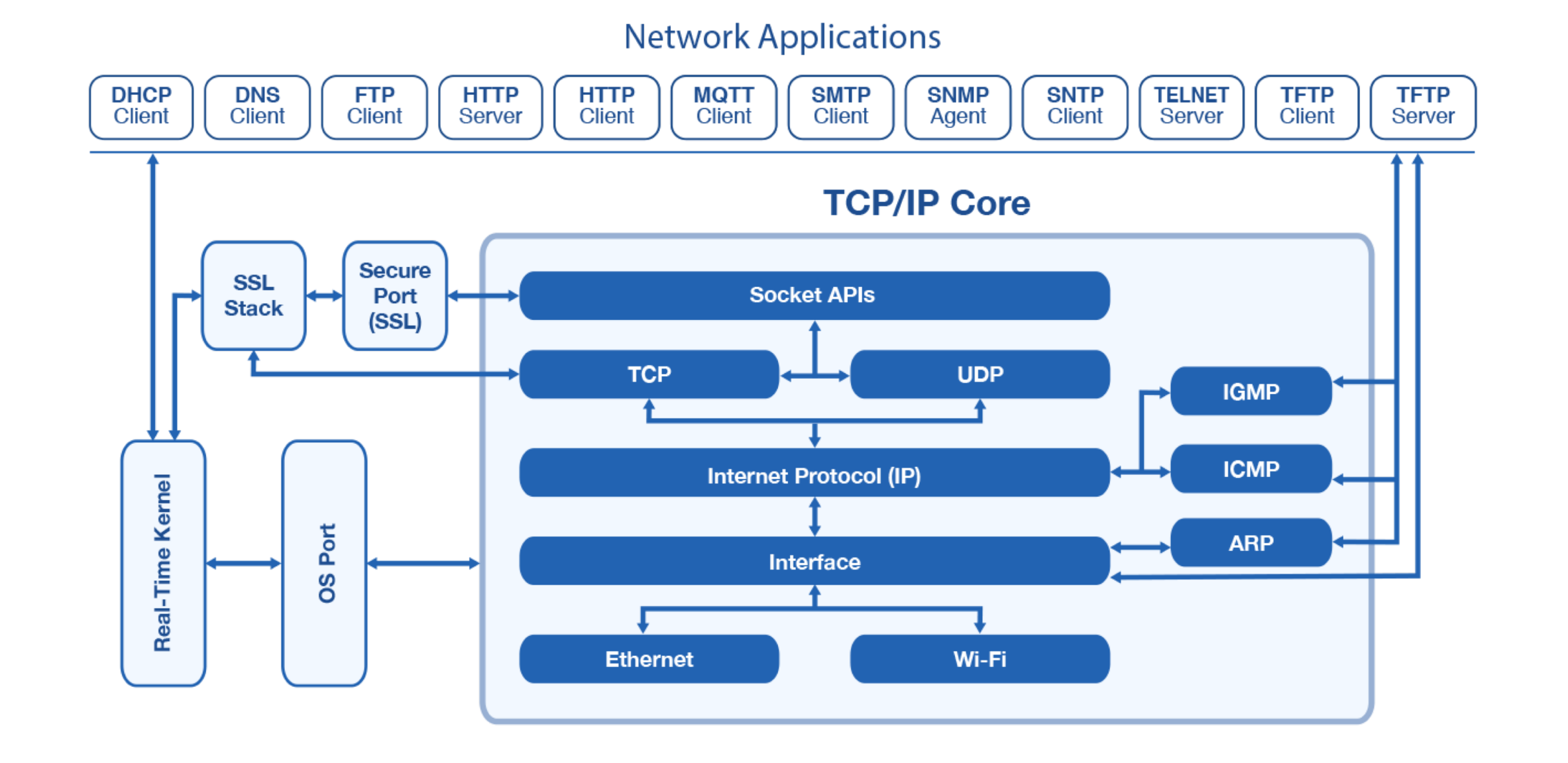
Micrium OS Network

Network History

- First version of µC/TCP-IP released in 2002
  - One of the first non-kernel modules released by Micrium
- Book released in 2009
  - Accompanied the release of the USB book
- IPv6 support added in 2015
- Micrium OS released in 2017
  - µC/TCP-IP becomes the Micrium OS Network component
  - Network stack core moves from three tasks to one
  - DHCP refactored
  - Network interface setup simplified



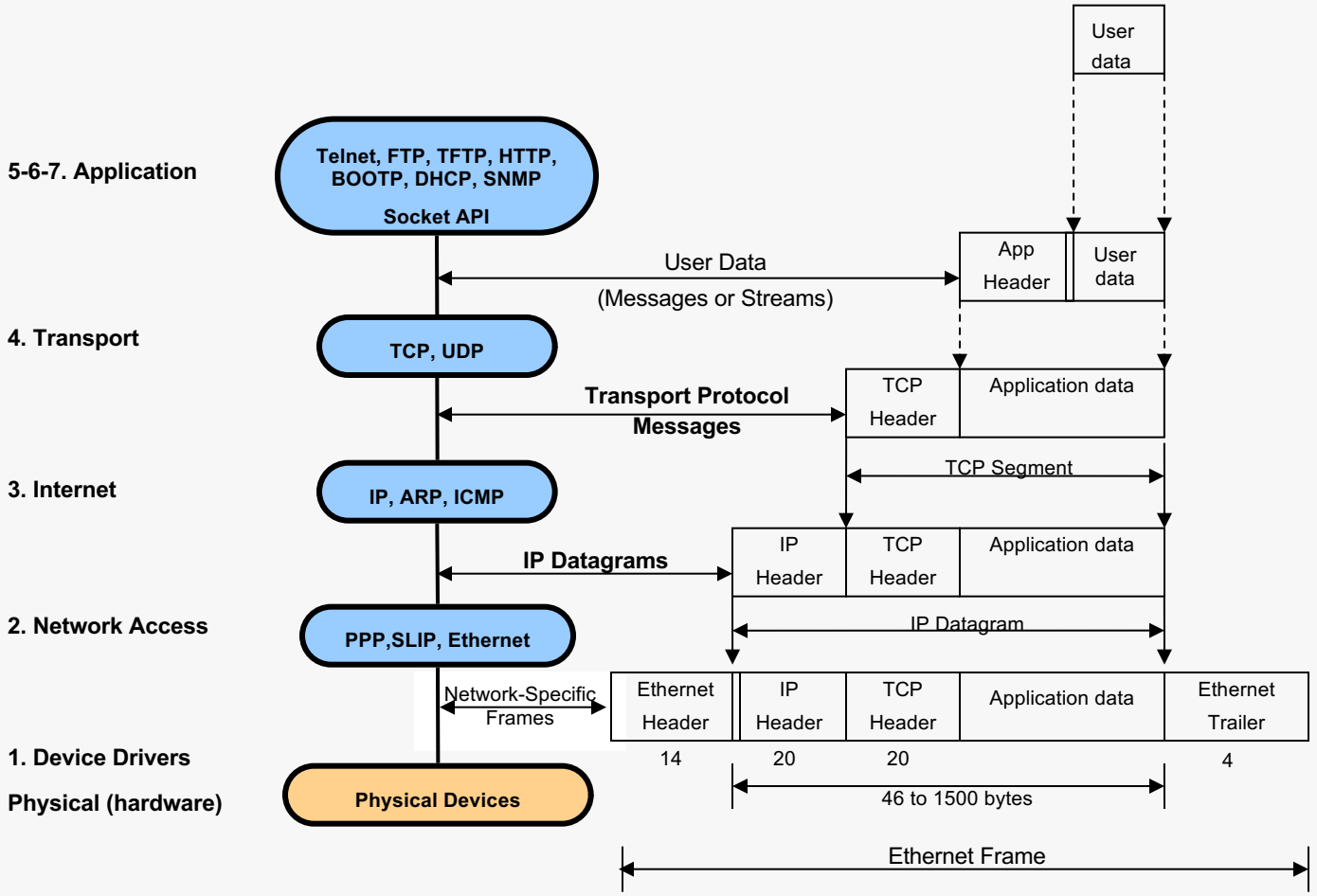
Network Layout



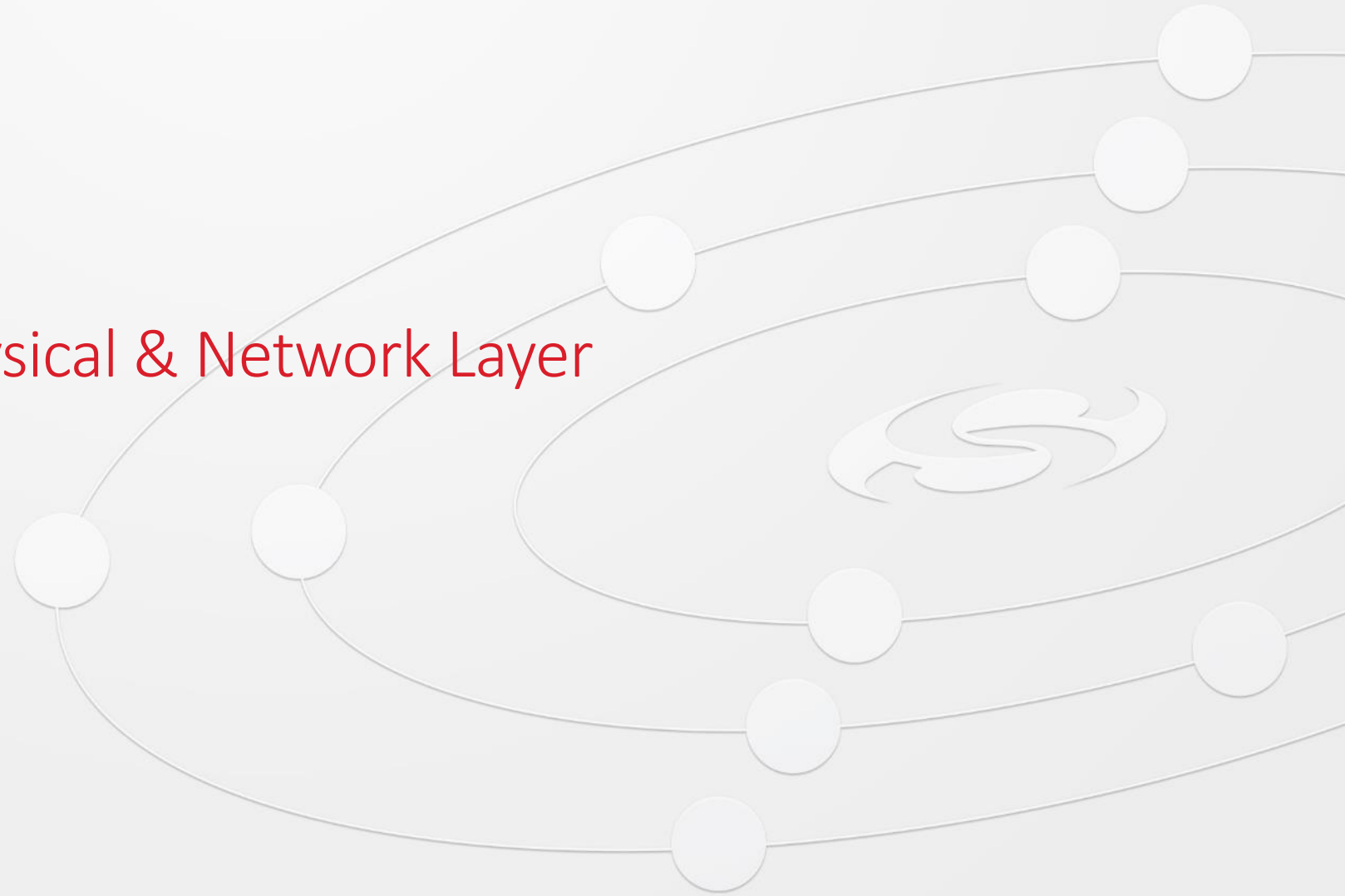
Network Features

- Micrium's TCP/IP stack offers the following features: Support for multiple interfaces and multiple IP addresses per interface (multihoming)
  - Support for IPv4 and IPv6
  - A BSD socket API with most popular socket options
  - Scalable to contain only required features and minimize memory footprint
  - SSL/TLS support for secure sockets (sold separately)
  - Interface support: Ethernet, Wi-Fi, Loopback
- IPv6 Support
  - Simultaneous use of IPv4 and IPv6
  - IPv6 Node
  - SLAAC (IPv6 stateless address autoconfiguration)
  - IPv6 Multicast (MLD)
  - Neighbor Discovery Protocol (NDP)
  - ICMPv6 (echo request/response)

OSI Model and TCP/IP



Micrium OS Network Physical & Network Layer



Physical Layer

- Defines the medium used in transporting data
  - Twisted-pair copper wire, fiber optic cable, etc.
- Not entirely a hardware concern
- Often implemented through a PHY chip on embedded platforms
  - Examples: Marvell 88E1111, AMD AM79C874
- PHYs can communicate with link-layer hardware through any of a number of protocols
  - MII, RMII, GMII, RGMII

Physical Layer in Micrium OS Network

- Small driver required for PHY chips
  - ~1000 lines of code
- For many parts, generic driver is acceptable
  - Micrium already has drivers developed for most popular PHYs
- Debugging PHY driver can be tricky
  - Documentation for hardware might not be readily available
- The structure `NET_PHY_CFG_ETHER` is used to set the PHY configuration in the stack
  - Examples of this can be found in the network BSP for your specific board
  - Sets the PHY Bus Address, PHY Bus Mode, PHY type, link speed and duplex type.

Network Layer

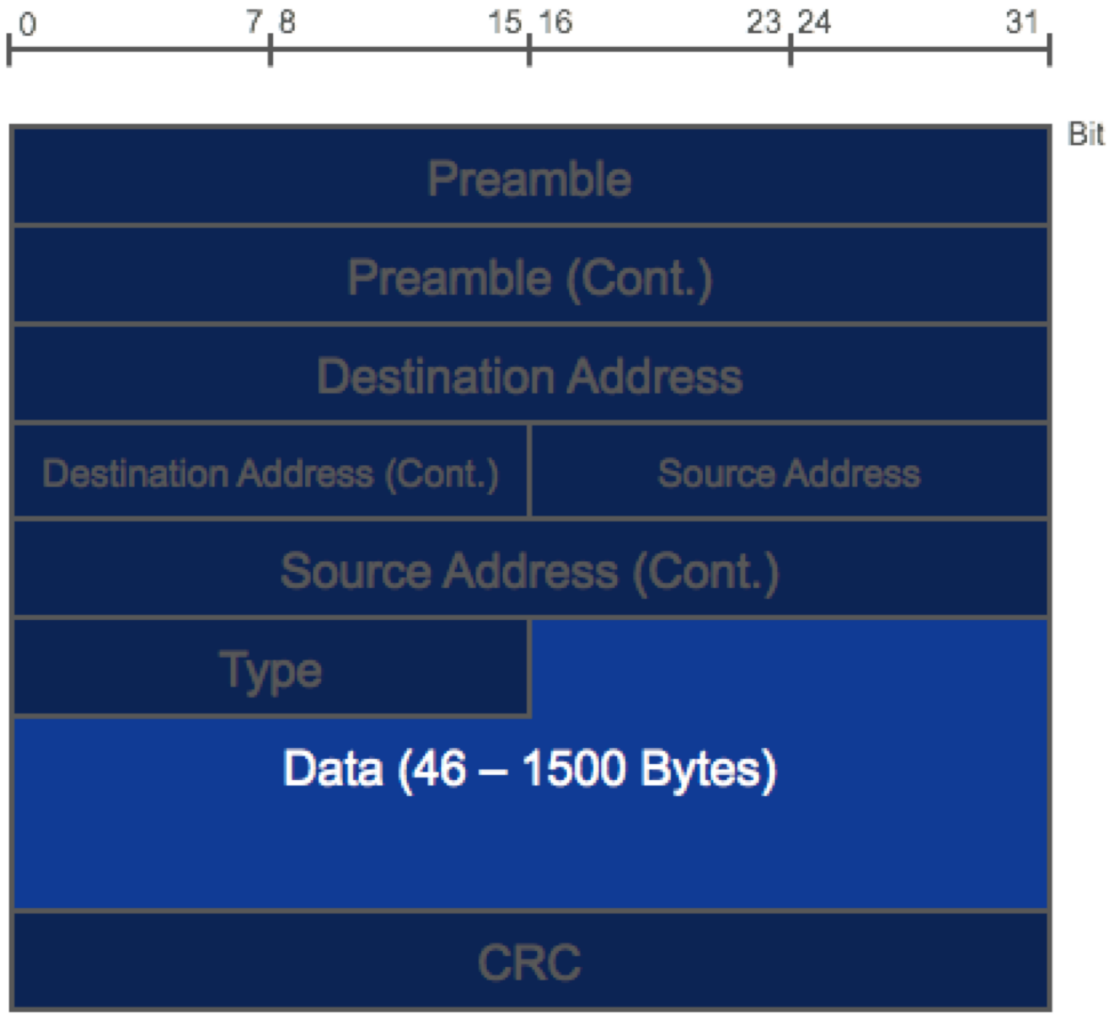
- Establish communications link between two or more hosts
- Services provided *may* include framing, reliable delivery, flow control, and error detection
- Data is delivered via frames to other devices on the same Local Area Network
- Protocol examples at the Network Layer:
  - Ethernet
  - IEEE 802.11 (WiFi)
  - Point-to-Point (PPP)
  - I2C
  - 1-Wire

Ethernet Addresses and Speed

- Every device on an Ethernet-based LAN must have an address
  - MAC address
- Address is 6 bytes long and is unique
  - IEEE manages addresses
- Special addresses
  - Broadcast: FF:FF:FF:FF:FF:FF
  - Multicast: First bit of leftmost byte set

- Speed
  - IEEE 802.3: 10 Mbps
  - IEEE 802.3u: 100 Mbps
  - IEEE 802.3z: 1,000 Mbps
  - Speed of a particular link often determined through *auto-negotiation*

Ethernet Frame



Network Layer in Micrium OS

- Requires a driver to be written for the specific network controller
  - Micrium has drivers written for some of the more popular network controllers
  - Existing drivers can be found in rtos/drivers/net
  - Not a trivial task to write a network driver
- The structure NET_DEV_CFG_ETHER contains your configuration for items such as
  - Rx and Tx buffer quantities
  - Rx and Tx buffer sizes
  - Dedicated memory pools
  - Base Ethernet controller address
  - MAC address value
- An example of how to configure the structure can be found in the network BSP for your board

Initializing the network stack

- `Net_Init()`: Initialize the core network module and task
- `NetIF_XXX_Add()`: Adds an Ethernet/WiFi interface to the network module
- `NetIF_XXX_Start()`: Starts the interface by setting up the controller for Ethernet/WIFI. Also may set the MAC address, IP address or enable DHCP.

- Examples for starting Ethernet/WIFI as well as all of the network modules can be found in the examples folder.

Establishing a Connection

- After initializing the resources for your TCP-IP Network you will need to try and establish a connection with your network interface (Ethernet/WiFi)
- `NetIF_LinkStateWaitUntilUp()`
  - Allows the user to specify the number of times to check the state, and the delay time
- `NetIF_LinkStateSubscribe()`
  - Allows the user to create a callback function for whenever the links state changes (UP or DOWN)

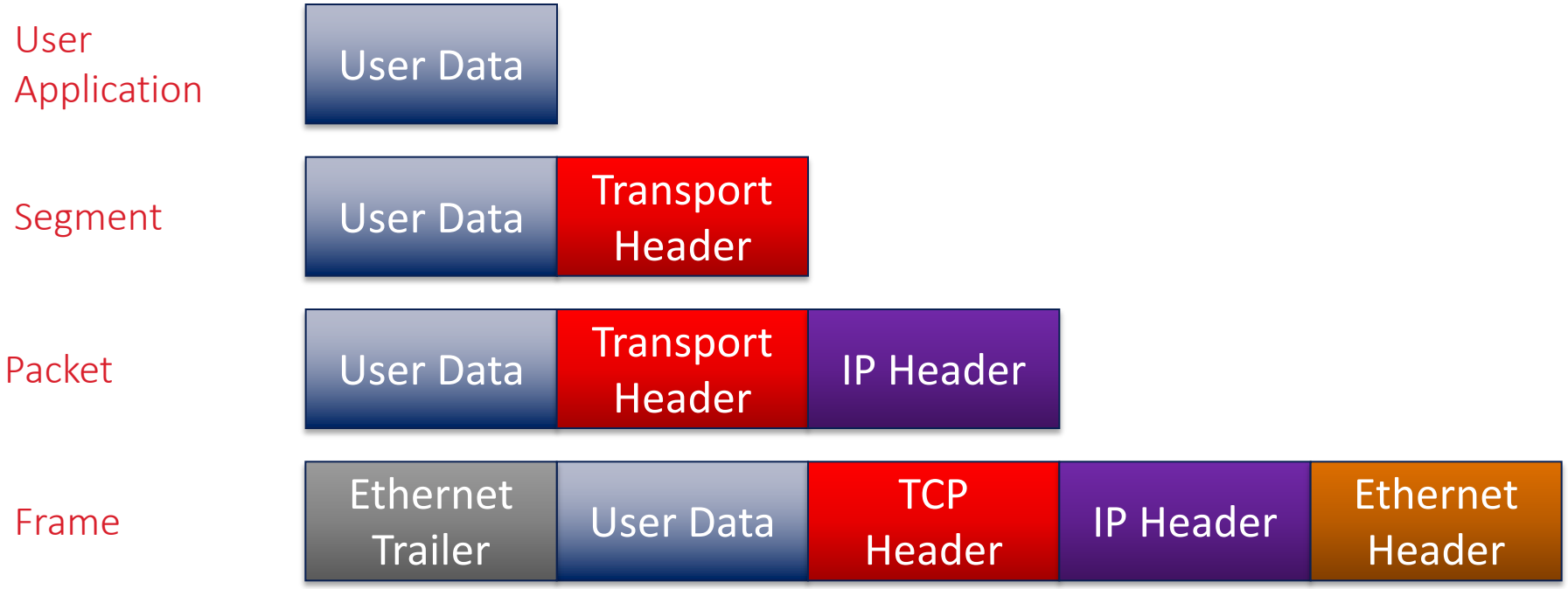
Micrium OS Network IP Layer

IP Basics

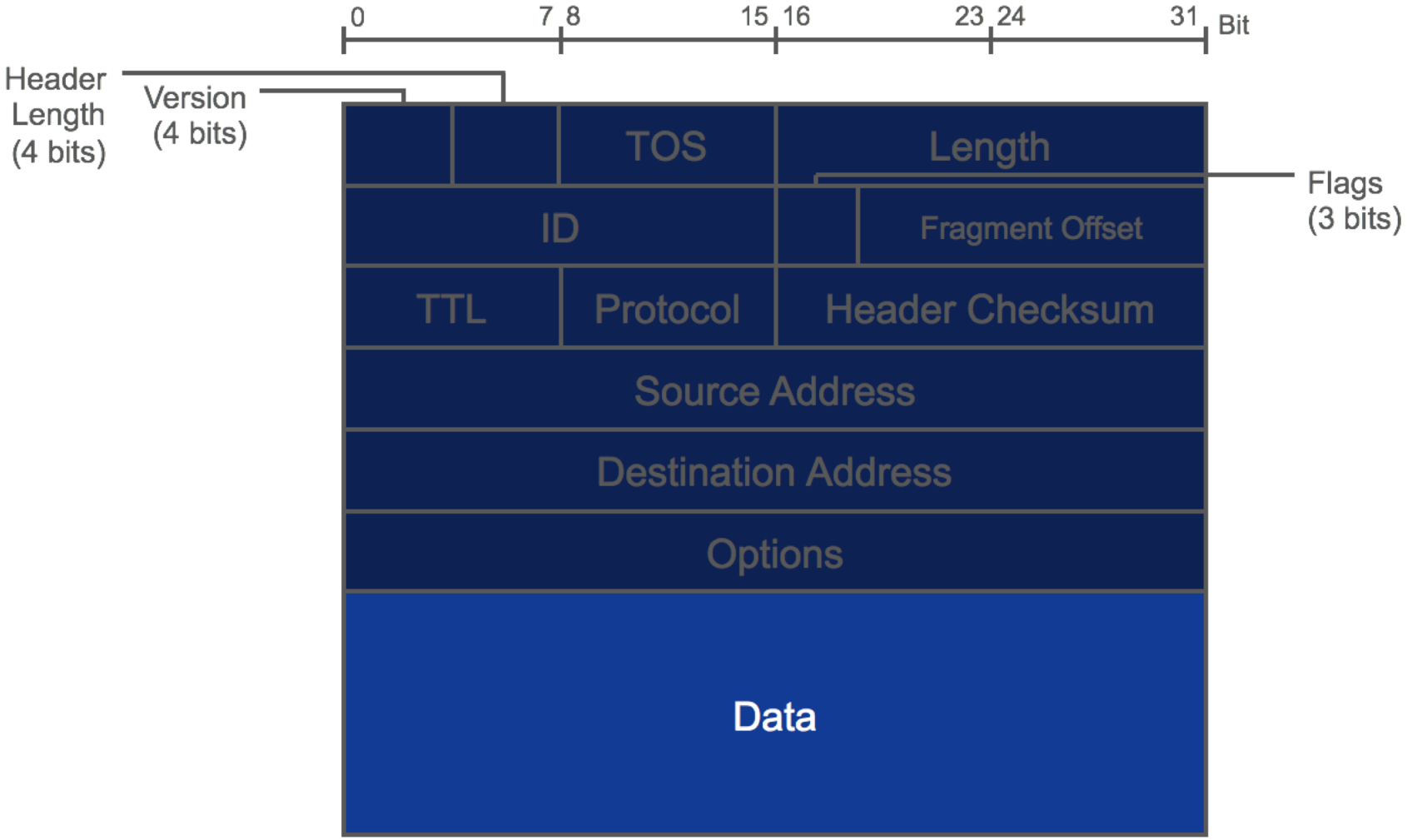
- Protocol intended to connect disparate LANs (Local Area Networks) built on link-layer technology
- Provides *best-effort* service that does not ensure data delivery
- Each host in an IPv4 network has a 32-bit address
  - Dotted-decimal notation: 153.215.24.116
- Addresses managed by ICANN (Internet Corporation for Assigned Names and Numbers)
- 2 portions of every address: subnet and host address

Data Encapsulation

- On an Ethernet LAN, every IP datagram must be encapsulated in a frame
  - The Frame is made up of several different layers
    - **Frame, Packet, Segment** and **User Data**



Generic Datagram Format



Getting an IP Address

- DHCP
  - If the DHCP module is enabled, Micrium OS Network will default to attempting to pull an IP address from a DHCP server
    - If there is no DHCP server available, or DHCP is not enabled, Micrium OS Network will fall back to the default Static IP address value.

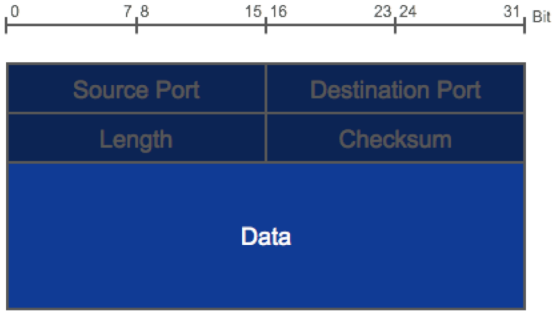
- Static Address
  - Functions `NetIPv4_CfgAddrAdd` and `NetIPv6_CfgAddrAdd` can be used to configure a static address value
    - These functions require the use of `NetASCII_Str_to_IPv4()` and `NetASCII_Str_to_IPv6()` to convert an IP address string to the correct format for the `CfgAddrAdd`functions
  - If using the provided example code, a structure `Ex_NetIF_CfgDflt_Ether` is used for default values for IPv4, IPv6 or both depending on how the network stack is configured

UDP & TCP

UDP Basics

- Connectionless protocol
  - No state information maintained on either side of connection
  - Lightweight, no “congestion control”
  - Data is transmitted from the source to the destination without verify the readiness or state of the receiver
  - Has the ability to broadcast to all devices on the subnet
- Datagrams are not guaranteed to reach their destination
  - Packets may arrive out of order to destination
- Provides checksums for data integrity
  - Calculation is different between IPv4 and IPv6
  - Is optional in IPv4
- Popular in applications requiring high throughput
  - Used by DNS, SNMP, RIP and DHCP
  - Media/Video Streaming (lost frames are acceptable)
  - Video Games not concerned with 100% data accuracy

UDP Datagram Format



UDP (Datagram) Socket Functions

- `NetSock_Open()` ↔ `socket()`
  - sets up communication endpoint
- `NetSock_Bind()` ↔ `bind()`
  - associates a port number with a socket
- `NetSock_TxDataTo()` ↔ `sendto()`
  - send data to a particular machine (IP address)
- `NetSock_RxDataFrom()` ↔ `recvfrom()`
  - receive data from specified source

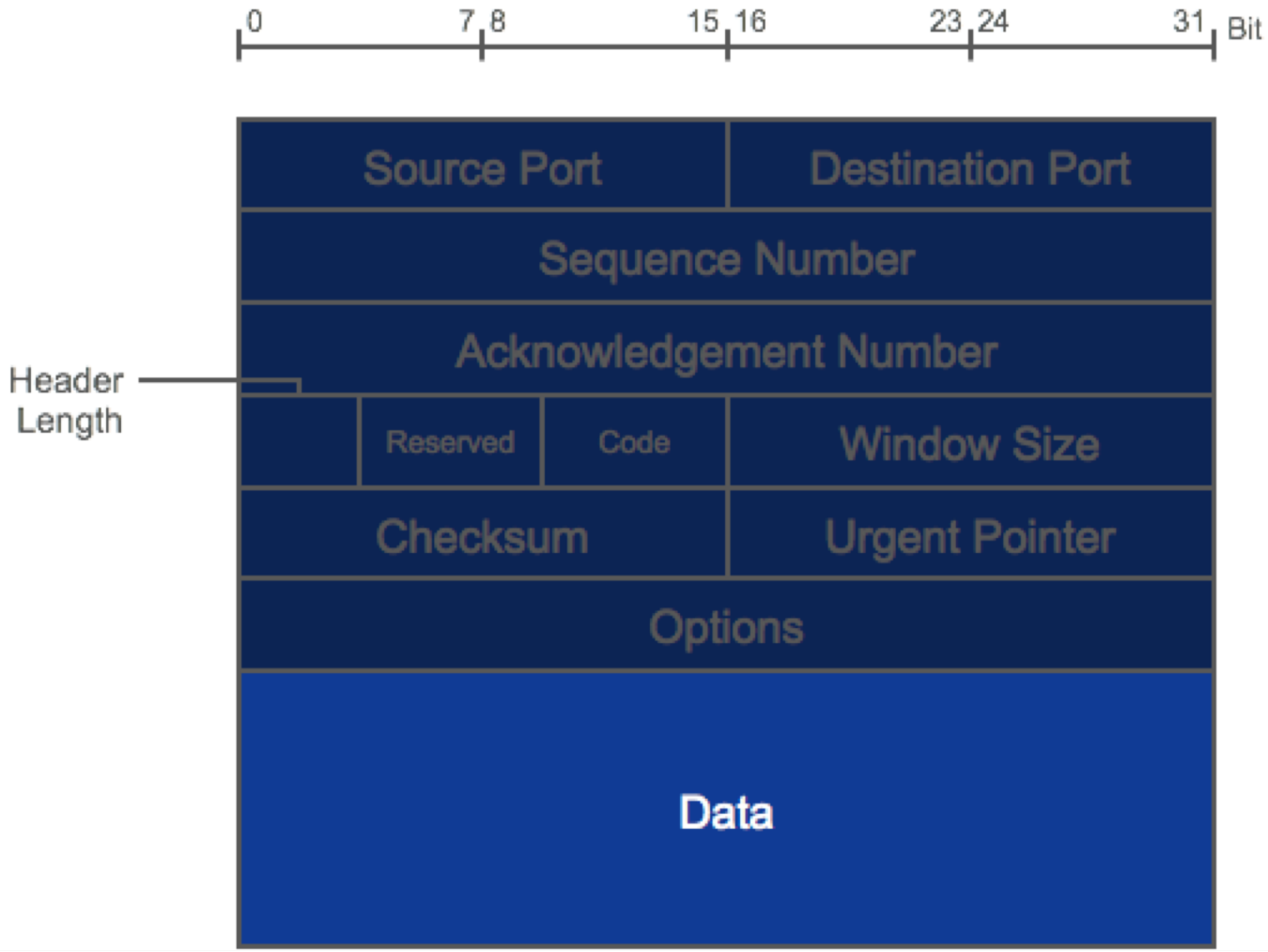
TCP Basics

- A connection-oriented protocol with state information maintained on each communicating device
  - More overhead involved in data transmission
- Reliable, in-order data transfer
  - Buffering and queueing used to ensure data integrity
- Error checked
  - 16-bit checksum that verifies:
    - Header
    - Payload
    - Source IP
    - Destination IP
    - Protocol number
    - Length of TCP header
- Used in applications where data integrity and accuracy is critical
  - Web messaging applications
  - SSH, TELNET, FTP, SMTP
  - E-Mail services

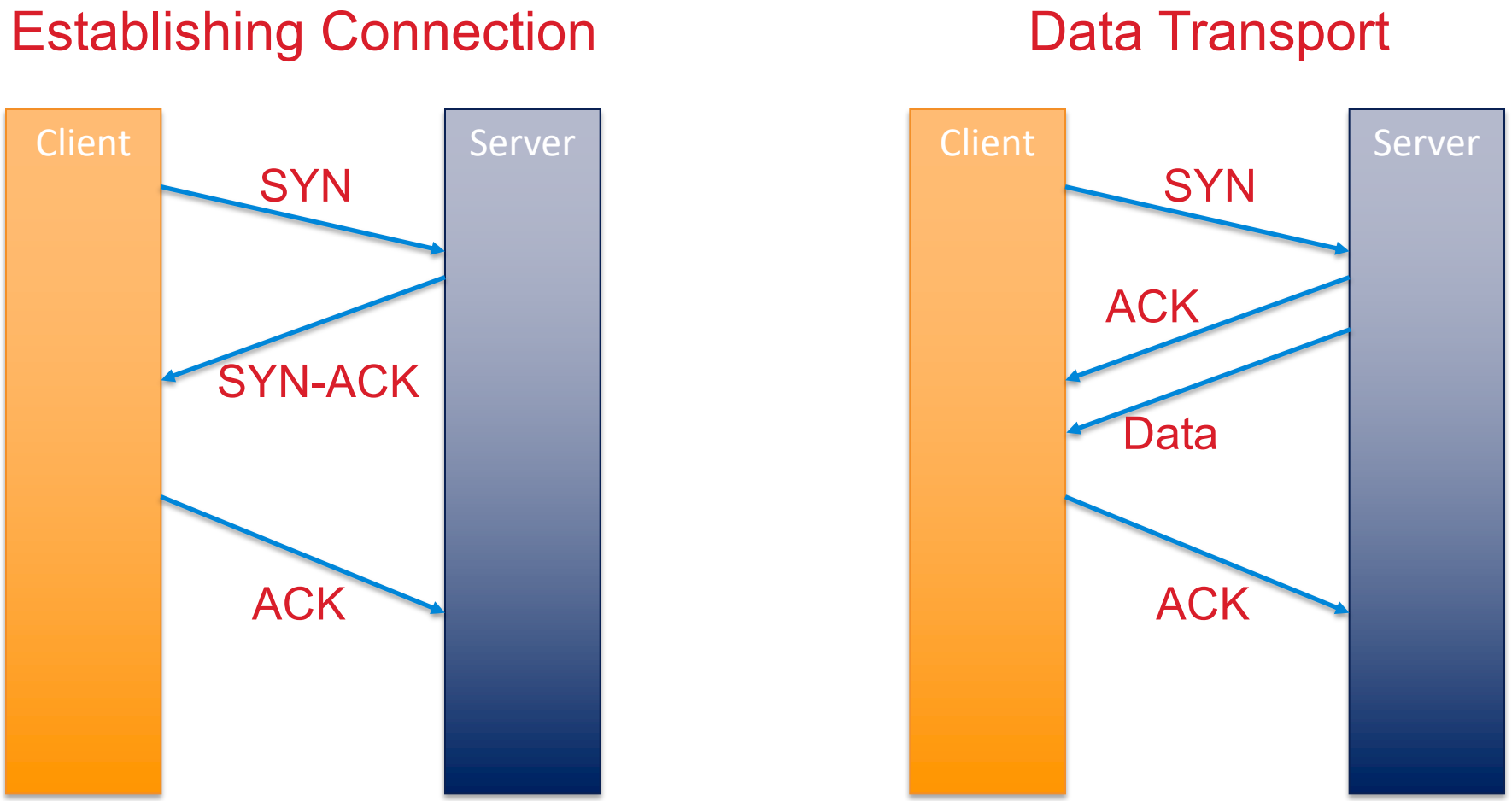
TCP Advanced Features

- Window *Flow control* to ensure devices aren't sent more data than they're able to receive
  - User can configure window sizes for TCP applications
  - Allows for a "smarter" system to efficiently handle incoming data
- *Congestion control* prevents a device from clogging the network with packets
  - Congestion causes a system to lose packets, trigger retransmissions, slow data throughput and even system crashes
  - Proper window size configuration is essential to prevent congestion

TCP Segment Format



Typical TCP Communication Sequence



TCP (Stream) Socket Functions

- `NetSock_Conn()` ↔ `connect()`
  - establish a connection with a remote host
- `NetSock_Listen()` ↔ `listen()`
  - wait for an attempt to connect
- `NetSock_Accept()` ↔ `accept()`
  - returns new socket corresponding to connection
- `NetSock_TxData()` ↔ `send()`
  - send data to connected host
- `NetSock_RxData()` ↔ `recv()`
  - receive data from connected host

Micrium OS Network UDP Lab Overview

UDP Client Socket Example

- `NetSock_Open()`: Opens a network socket. Specify IPv4 & UDP.
- `NetApp_SetSockAddr()`: Sets up the socket address and specifies the port
- `NetSock_Conn()`: Connects the socket to the remote host

- `NetSock_TxDataTo()`: Transmits data via the socket to the remote host

UDP Server Socket Example

- `NetSock_Open()`: Opens a network socket. Specify IPv4 & UDP.
- `NetApp_SetSockAddr()`: Sets up the socket address and specifies the port
- `NetSock_Bind()`: Binds the socket to a local address

- `NetSock_RxDataFrom()`: Receives data from the remote connection

Micrium OS Network UDP Lab

Client

Server

BTN0 → **LED0**

BTN1 → **LED1**

Micrium OS Network Applications

DHCP Client

- Highlights
  - Transmit and receive option supported
  - IPv4 Only

- Usage
  - Requires `NET_DHCP_CLIENT_CFG_MODULE_EN` to be set in net_cfg.h
  - Does not require any direct call made to the DHCP module
    - By enabling `NET_DHCP_CLIENT_CFG_MODULE_EN`, the network service task initializes DHCP for you

DNS Client

- Highlights
  - IPv4 and IPv6 support
  - Auto selects the best address
  - Integrated cache to prevent multiple lookups of the same address

- Usage
  - Requires `NET_DNS_CLIENT_CFG_MODULE_EN` to be set in net_cfg.h
  - Initialization calls to the DNS module are automatically made from the net services task
  - To convert a string representation of a host name to its IP address, make a call to `DNSc_GetHost()`

FTP Client

- Highlights
  - Supports username/password authentication
  - Transfer via buffers or files
  - Supports SSL/TLS

- Usage
  - Requires `RTOS_MODULE_NET_FTP_CLIENT_AVAIL` to be set in rtos_description.h
  - Connections to open a remote host are made via `FTPc_Open()`
  - If transferring files via a buffer, `FTPc_RecvBuf` and `FTPc_SendBuf` are used
  - If transferring files via a file system, `FTPc_RecvFile` and `FTPc_SendFile` are used

MQTT Client

- Highlights
  - Implements MQTT 3.1.1
  - Supports multiple broker connections and multiple messages in transmit
  - Allows for an unlimited number of connections and messages
  - Support for all Qualities of Services (QoS-0, QoS-1, QoS-2)
  - Supports SSL/TLS

- Usage
  - Requires `RTOS_MODULE_NET_MQTT_CLIENT_AVAIL` to be set in rtos_description.h

Other Network Applications

- SMTP Client
  - Support for authentication and SSL/TLS
- SNTP Client
- Telnet Server
  - Integrated with Micrium OS Shell
  - Support authentication and SSL/TLS
- TFTP Client
  - Transfer and reception via buffers or files
- TFTP Server
  - Supports any file system; a file system is required

HTTP Client and Server

HTTP Protocol

- Hypertext Transfer Protocol
  - Application protocol that sits on top of the transport layer
  - Request-response protocol in a client-server model
- Originally developed by Tim Berners-Lee at CERN in 1989
- Protocol is described in various RFCs (request for comments)
- Has had four versions released
  - 1991 – HTTP/0.9
  - 1996 – HTTP/1.0
  - 1997 – HTTP/1.1
  - 2015 – HTTP/2.0

HTTP Client

- Full support for HTTP 1.1 (RFC 2616)
- All HTTP methods suported (GET, POST, PUT, DELETE, HEAD, OPTIONS, and TRACE)
- Support for WebSocket
- Support for persistent connections
- HTTP header field processing (HTTP cookies)
- Support multiple simultaneous connection with multiple HTTP Server

- Usage
  - Requires `RTOS_MODULE_NET_HTTP_CLIENT_AVAIL` to be set in rtos_description.h

HTTP Server

- Supports multi connection - up to 255 client connections
- Supports any file system, or can be used via a static file system, which is provided
- Include add-on for advanced functionalities such as Authentication Module and REST Framework
- Customizable hook functions to access events such as:
  - Page load
  - POST
  - GET

- Usage
  - Requires `RTOS_MODULE_NET_HTTP_SERVER_AVAIL` to be set in rtos_description.h

HTTP Server Lab