TPF Users Group
Grapevine, Texas

IBM Software Group

TPF 4.1 Communications - TCP/IP Enhancements

Jamie Farmer
October 2004

AIM Core and Enterprise Solutions
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Agenda

- Four new enhancements
  - All of the enhancements are at PUT 19 level
    - TCP Delayed Acknowledgement Enhancement
    - More Read TCP Message API's
    - TCP Push (PSH) Flag Enhancement
    - 10 GbE OSA-Express Support
- Recent TCP/IP APARs of Interest

TCP Delayed Acknowledgement Enhancement: PJ29638

What is a TCP Delayed Acknowledgement?

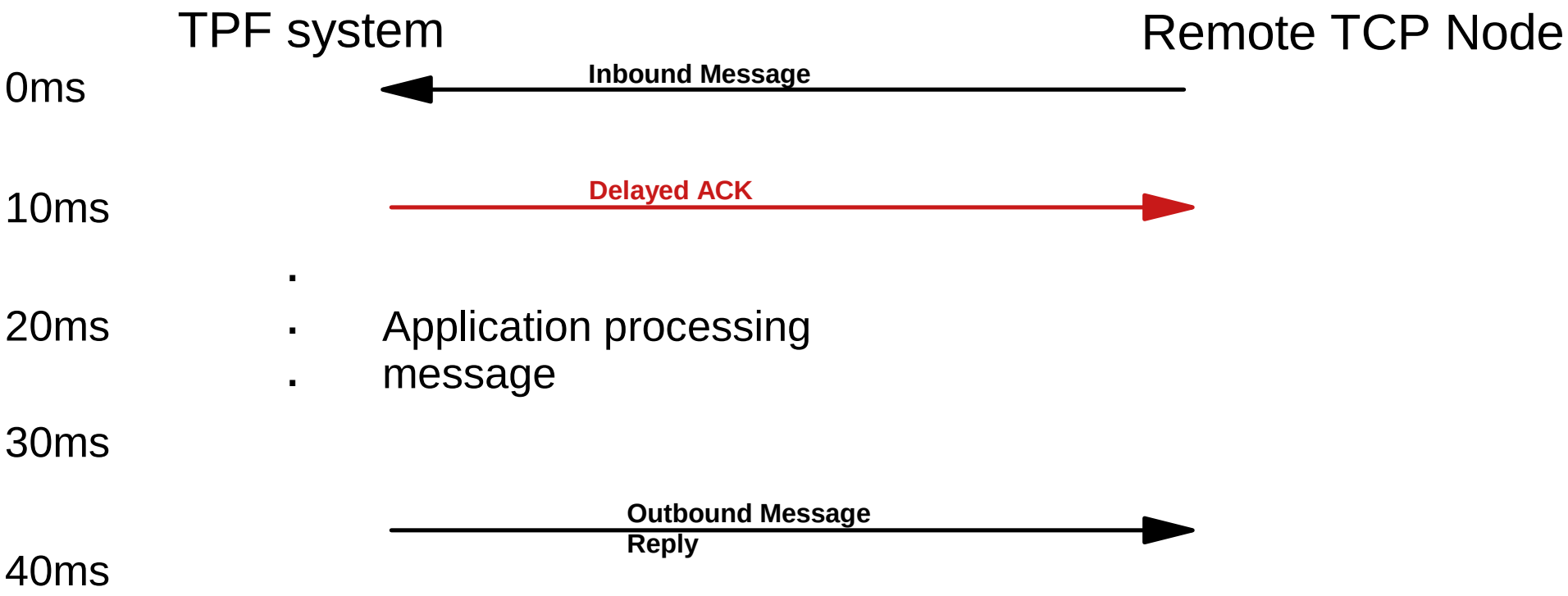
- Defines how long to wait when data arrives to send a standalone acknowledgment (ACK)
- If outbound data is sent before timer expires, the ACK will be piggybacked with the data.
  - This is optimal for performance
- Architecture allows delayed ACK timer values between 0-500 ms.
  - No architected way for applications to set or change the value
- Today, TPF uses a 10ms delayed ACK timer value.

PJ29638: TPF's alterable delayed ACK timer

- Ability to define the TPF system's delayed ACK timer
  - Alterable via ZNKEY using the TCPDELAY parameter
  - Valid range between 10-500ms, default is 10ms
- Ability to override the system's delayed ACK timer for a given socket
  - setsockopt() API option to modify timer for a given socket
    - Use the SO_TCPDELAY option
- Acknowledgements are sent if 2 full packets of data are received before the timer expires
  - The full packet size is based on the negotiated maximum segment size (MSS) for the socket.
  - Recommended by architecture

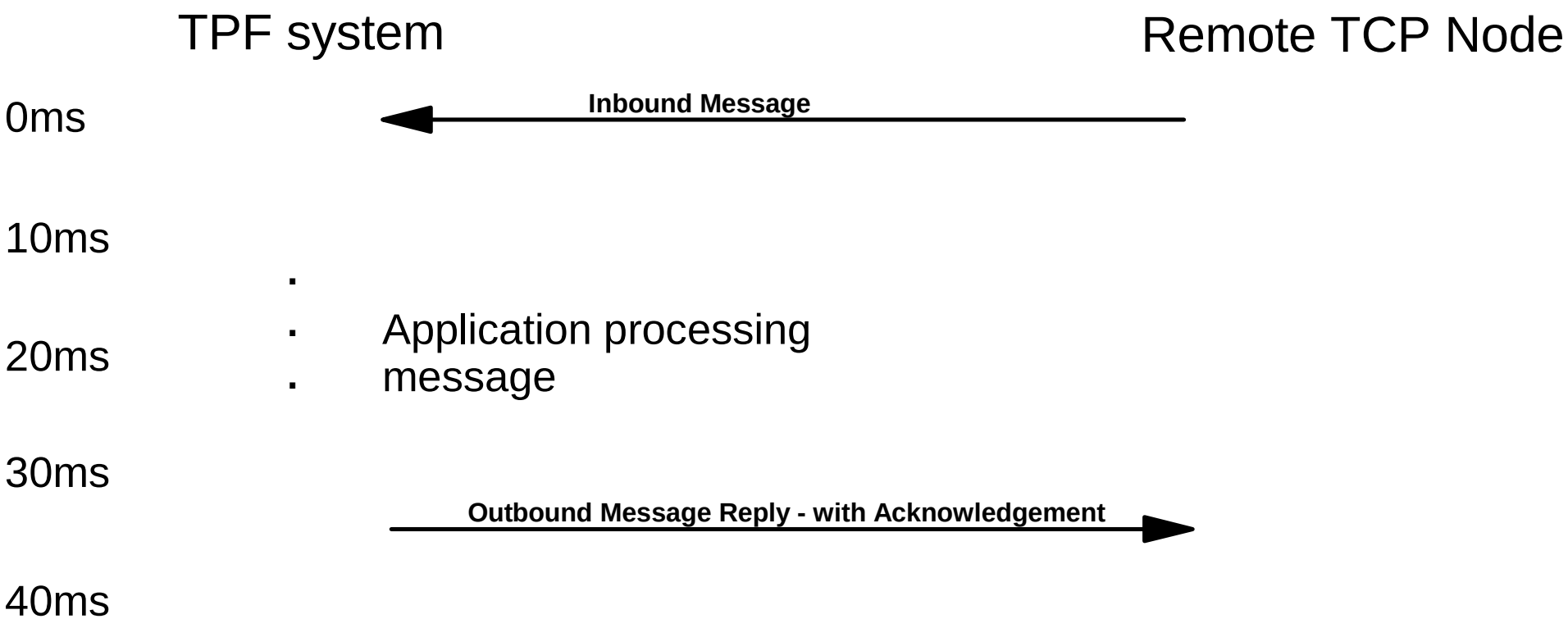
Why would I change a socket's delayed ACK timer value?

- To reduce the number of standalone ACK's sent by the system
  - Typical single threaded request-reply model application
    - Using a 10ms delayed ACK timer value
  - Takes application 30-40ms to process the message



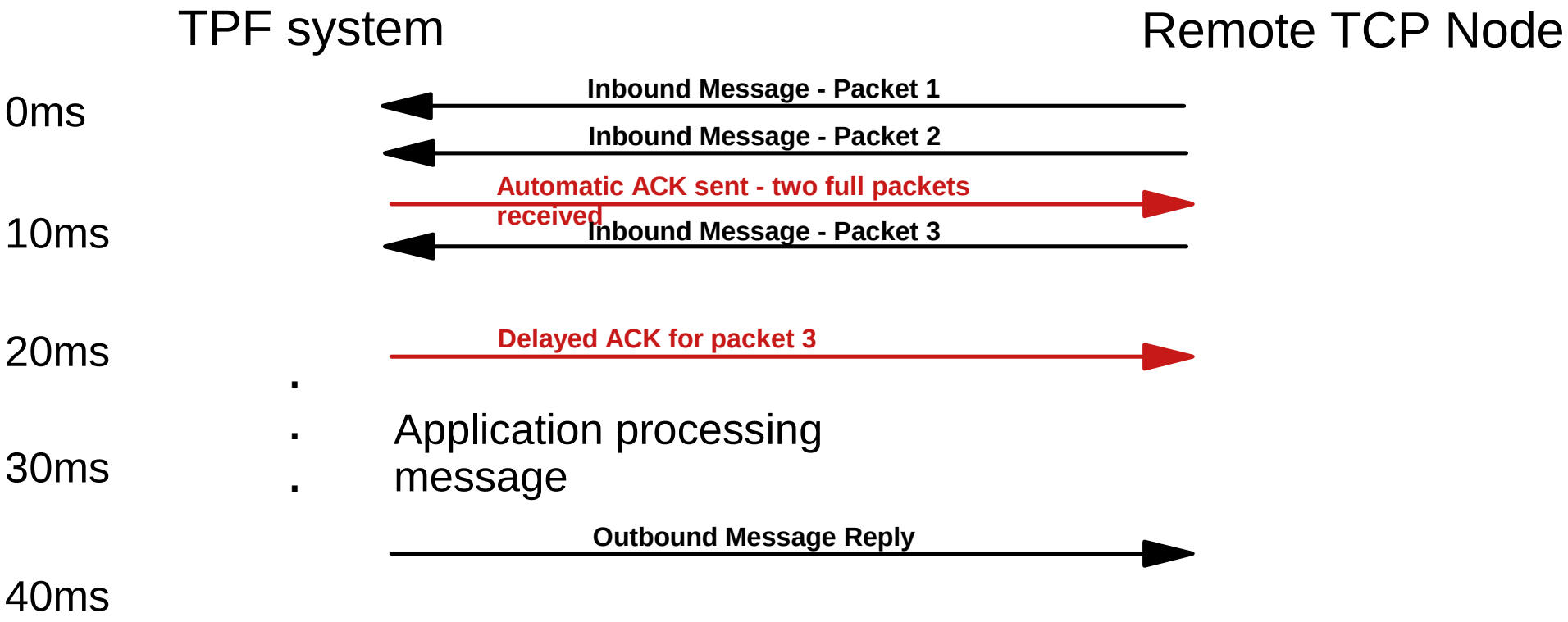
Example With an Increased Delayed ACK Timer Value

- Each inbound message is one packet of data
- Delayed ACK timer set at 40ms for this socket
- No standalone ACK sent in this example



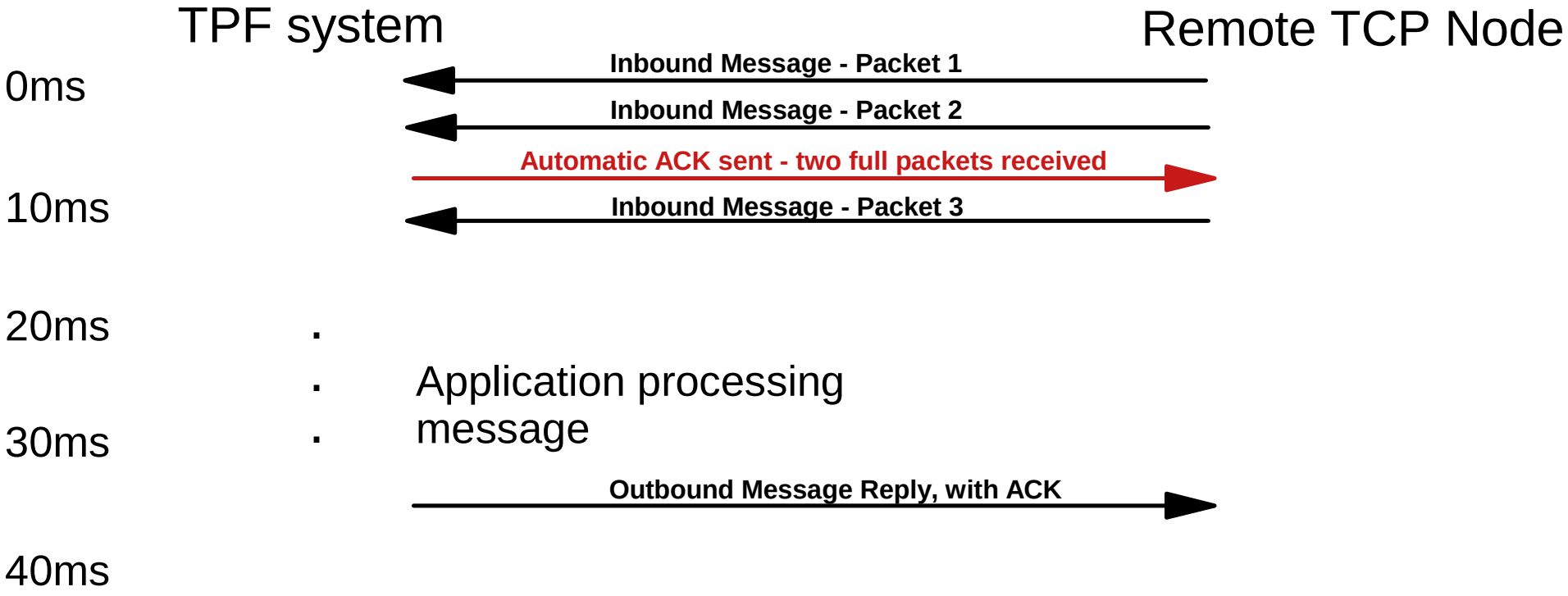
Example 2

- Typical single threaded request-reply model application
  - Using a 10ms delayed ACK timer value
- Each inbound message is three full packets of data
- Takes application 30-40ms to process the message



Example 2 with Increased Delayed ACK Timer

- Typical single threaded request-reply model application
  - Delayed ACK timer increased to 40ms.
- Each inbound message is three full packets of data
- Takes application 30-40ms to process the message
- Fewer standalone ACK's are sent.

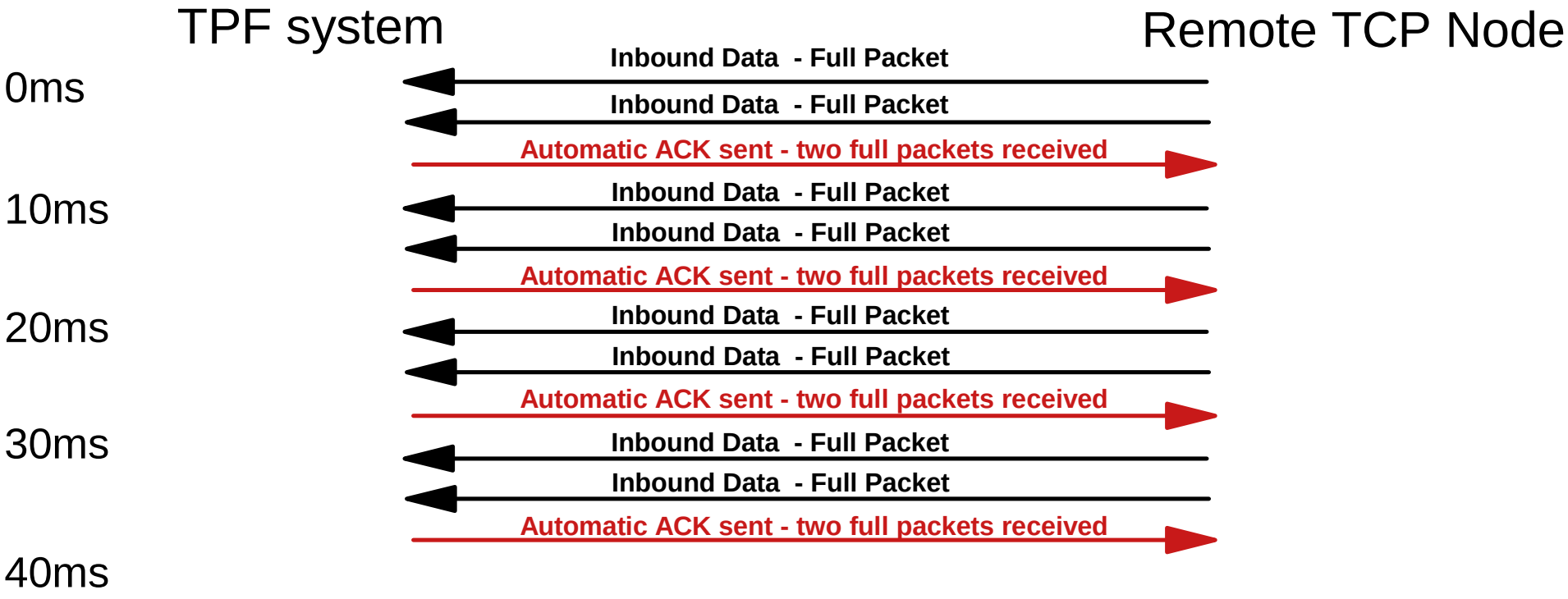


Why Not Increase the Delayed ACK Timer for All Sockets?

- Increasing the delayed ACK timer for certain sockets will have no effect on performance or throughput.
  - May even have some negative effects
- For example, let's take an example of a one-way pipe into TPF.
  - Due to an ACK being sent each time two full packets of data are received, increasing the delayed ACK value may have no effect.
- Another example is when the socket is a full-duplex socket.
  - Because we are constantly sending data outbound, we are piggy-backing all acknowledgements.
  - Standalone ACK's are unnecessary.

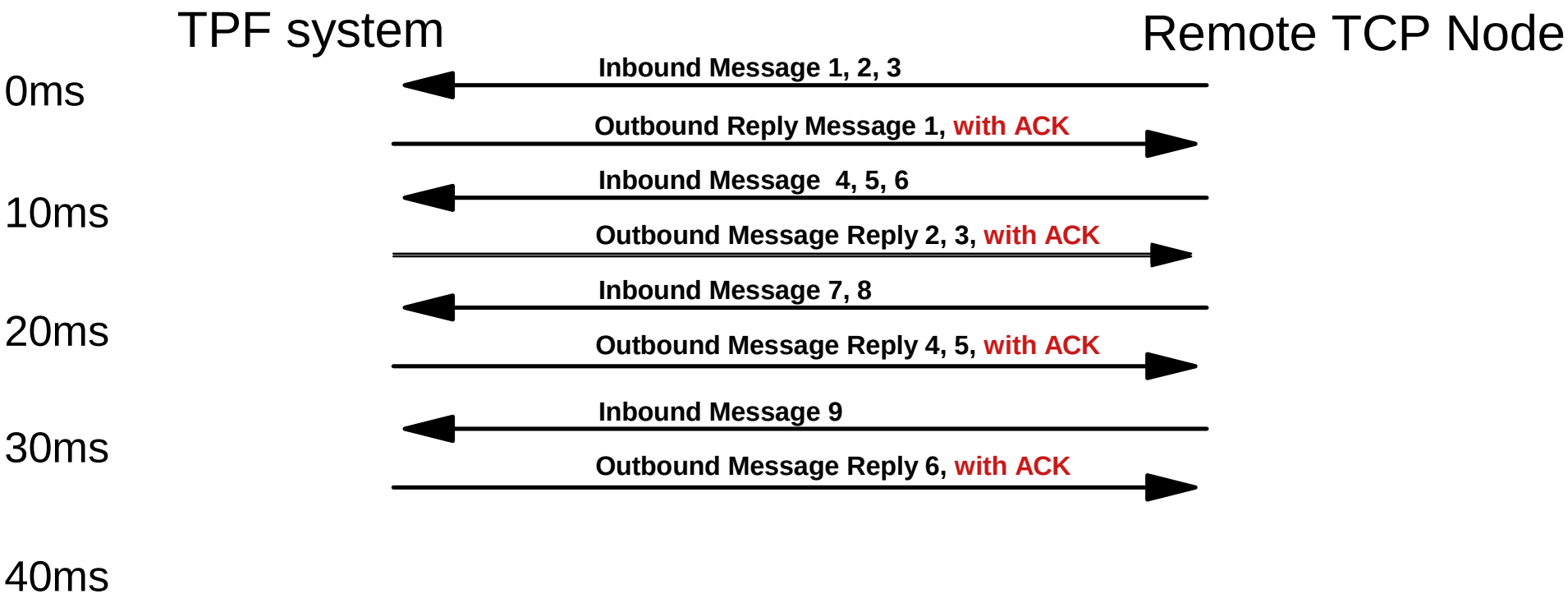
One-Way Pipe Into TPF

- With constant data flowing into TPF, the delayed ACK timer never expires
  - ACKs are sent because 2 full packets are received



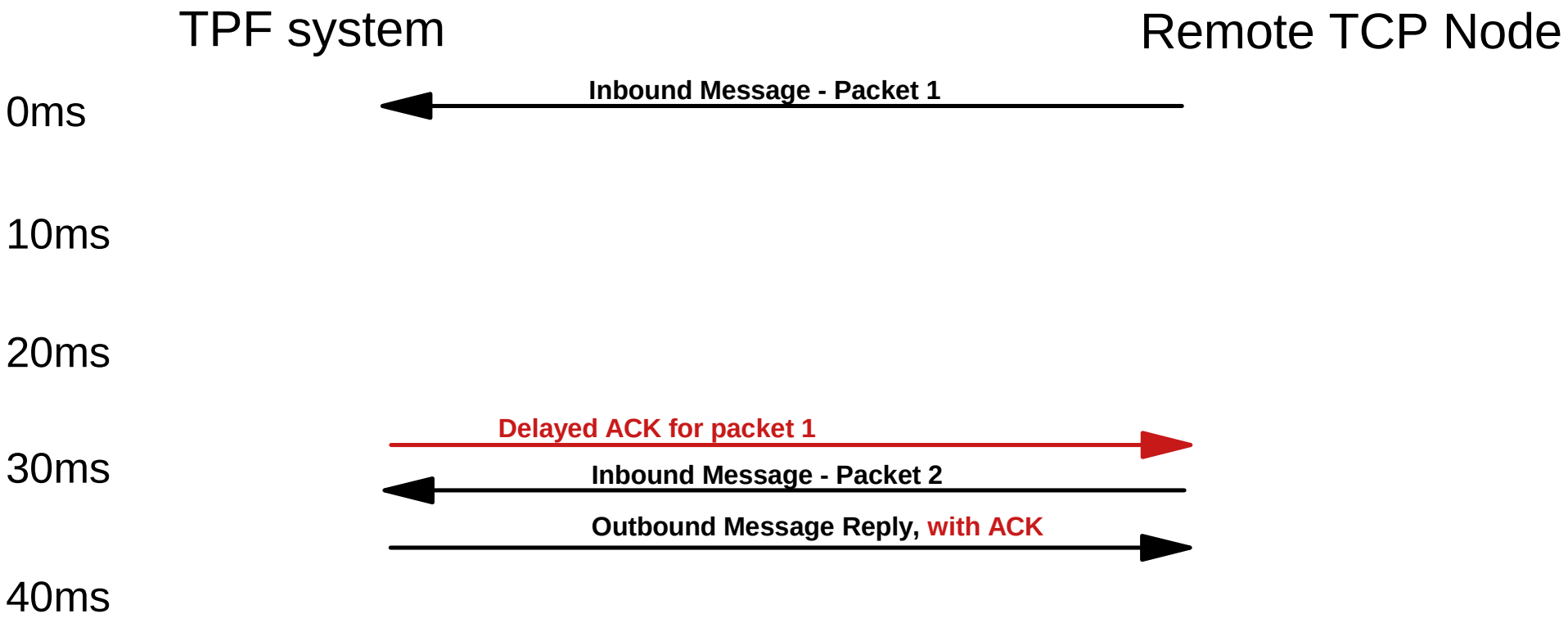
Full Duplex Socket

- With constant data flowing in and out of TPF, the delayed ACK timer never expires
  - ACKs are piggy-backed with outbound data



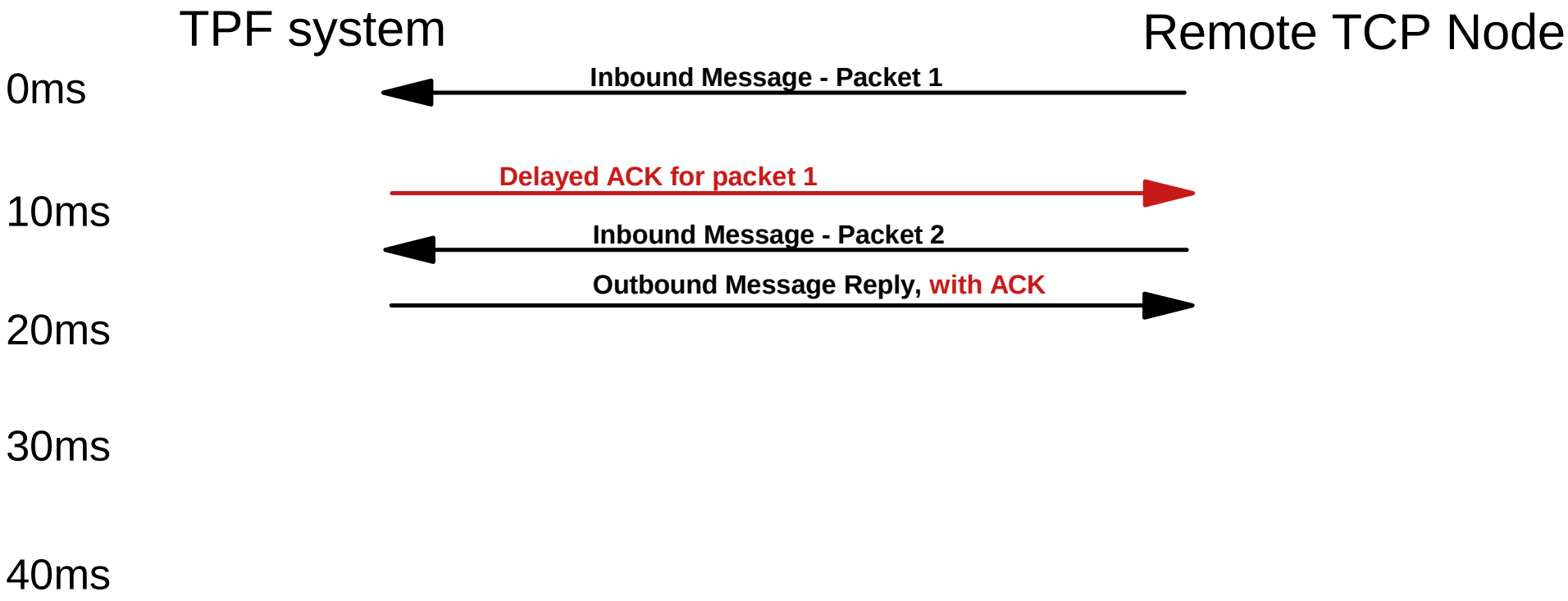
Example 3: Increasing the Timer Could Cause Problems

- Let's take an example of a short lived connection sending data into TPF
  - This remote node uses the standard slow start algorithm (part of congestion control and avoidance)
    - First packet's worth of data must be ACK'd by TPF before more data can be sent
- Using an increased delayed ACK timer of 30ms
- Each message is contained in two packets



Example 3 with Timer Set at 10ms

- Same example of a remote TCP node sending data into TPF
- Using default delayed ACK timer of 10ms
- Improved response time



Read TCP Message APIs
PJ29600

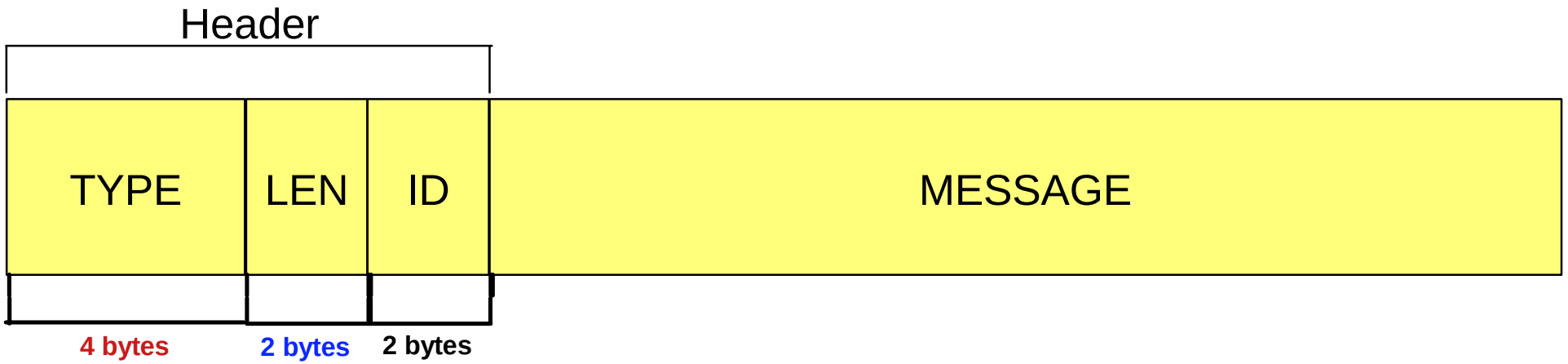
Reading TCP Message

- TCP architecture has no concept of a message
- Application data usually contains a header in front of the message containing the length of the message
- At least two reads for each message
  - One or more reads for the length
  - One or more reads for the message
  - For activate on receipt (AOR), multiple ECBs may be created for one message
- This has been a common cause of error in application programming
  - 4-Byte Header: A read for 4 bytes does not guarantee that 4 bytes will be received.

Original Read TCP Message APIs

- Through the Users Group we designed an API to read an entire TCP message based on the format of these messages
- APAR PJ29118
  - **tpf_read_TCP_message**
  - **activate_on_receipt_of_TCP_message**
- Performance improvement
  - Decrease in the number of APIs issued by the application to read a single message
  - Reduces the number of items dispatched on the CPU loop
  - Makes coding applications easier

Message Format of Original Read TCP Message API's



- In this example, header is 8-bytes with a 2-byte length field at offset 4
- Length field of message "includes" length of the header as well as the message size
- Two new parameters on the read message API
  - Offset of Length Field: **4**
  - Length of the length field: **2**

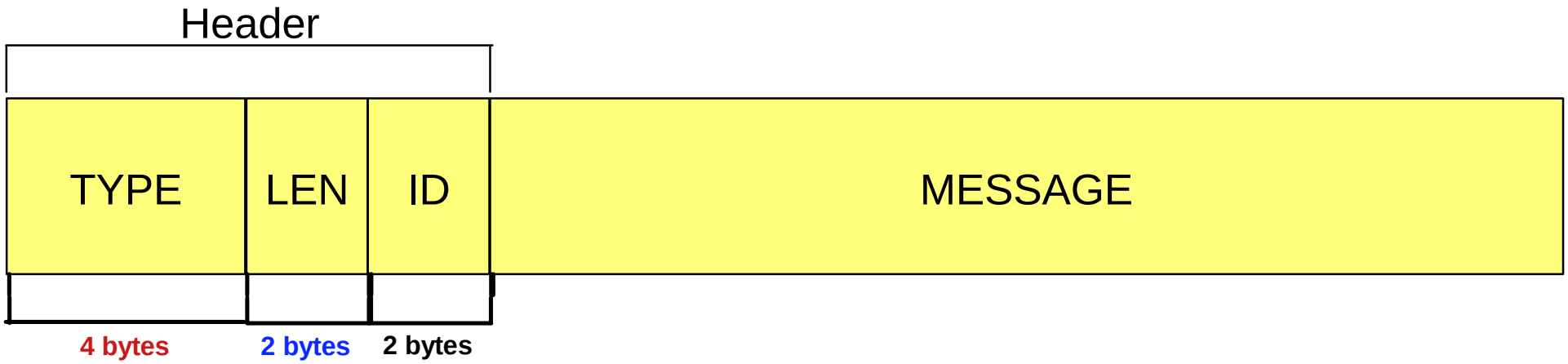
How the Read Message API Works

- The format of the message being received is passed as input to the API
  - Where the message length field within the header resides
    - The message length field does not have to be at the start of the header
  - The system will get the size of the message and read the entire message, including the header, before passing it to the application
    - The entire message, including the header, is passed to the application
    - Partial messages are never passed to the application

New Read TCP message APIs

- Customer requirement to support read message API's for another message format
- PJ29600 created two new APIs:
  - **tpf_read_TCP_message2**
  - **activate_on_receipt_of_TCP_message2**
- Functionally the same as the original APIs

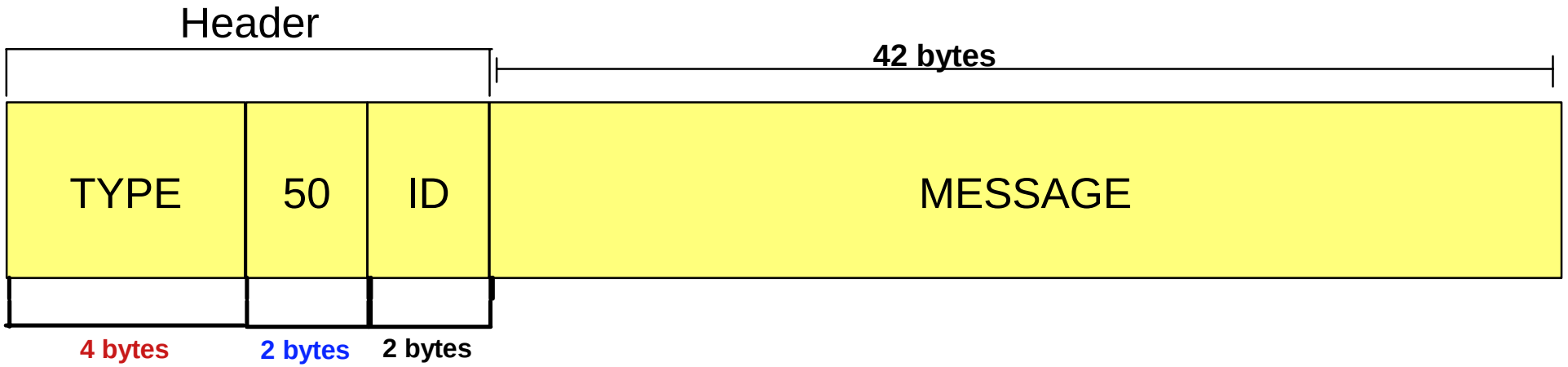
Message Format of New Read TCP Message API's



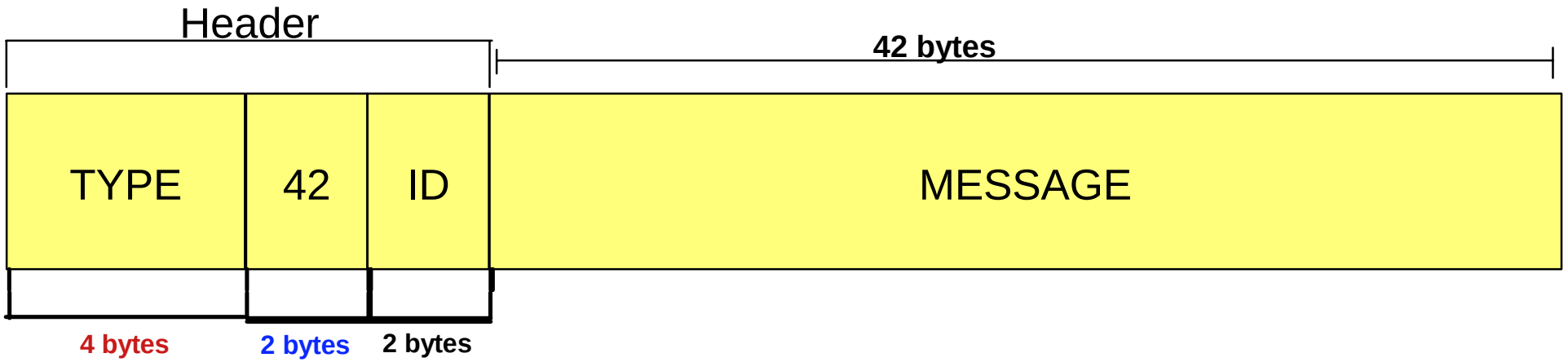
- In this example, header is 8-bytes with a 2-byte length field at offset 4
- Length field of message does **"NOT"** include header length
- Three new parameters on the read message API
  - Offset of Length Field: **4**
  - Length of the length field: **2**
  - Length of the entire header: **8**

Differences in Message Format

Original Format



New Format



TCP PSH Flag Enhancement PJ29832

What is the TCP Push (PSH) Flag?

- TCP architecture defines a PSH flag within the TCP header of the packet
- If the PSH flag is set on an inbound packet:
  - The TCP layer is supposed to immediately pass the data up to the application
- If the PSH flag is "NOT" set on an inbound packet:
  - The TCP layer may queue the data internally until a packet with the PSH flag is received.
  - The TCP layer may pass the data to the application
- A given TCP node's behavior is not necessarily consistent
  - For example, a node might queue the data if buffer space is available, otherwise pass data immediately to application

PSH Flag is not a Message Boundary

- TCP architecture clearly states that the PSH flag does not signify the end of a message.
  - TCP has no concept of a message
- An application can include multiple messages with one socket send type API.
  - Common practice with offload support where each send involves an I/O to the offload device.
- Socket send API could contain only a partial message
- **A TCP application cannot assume one read API will result in exactly one message**

Issues with the Architecture

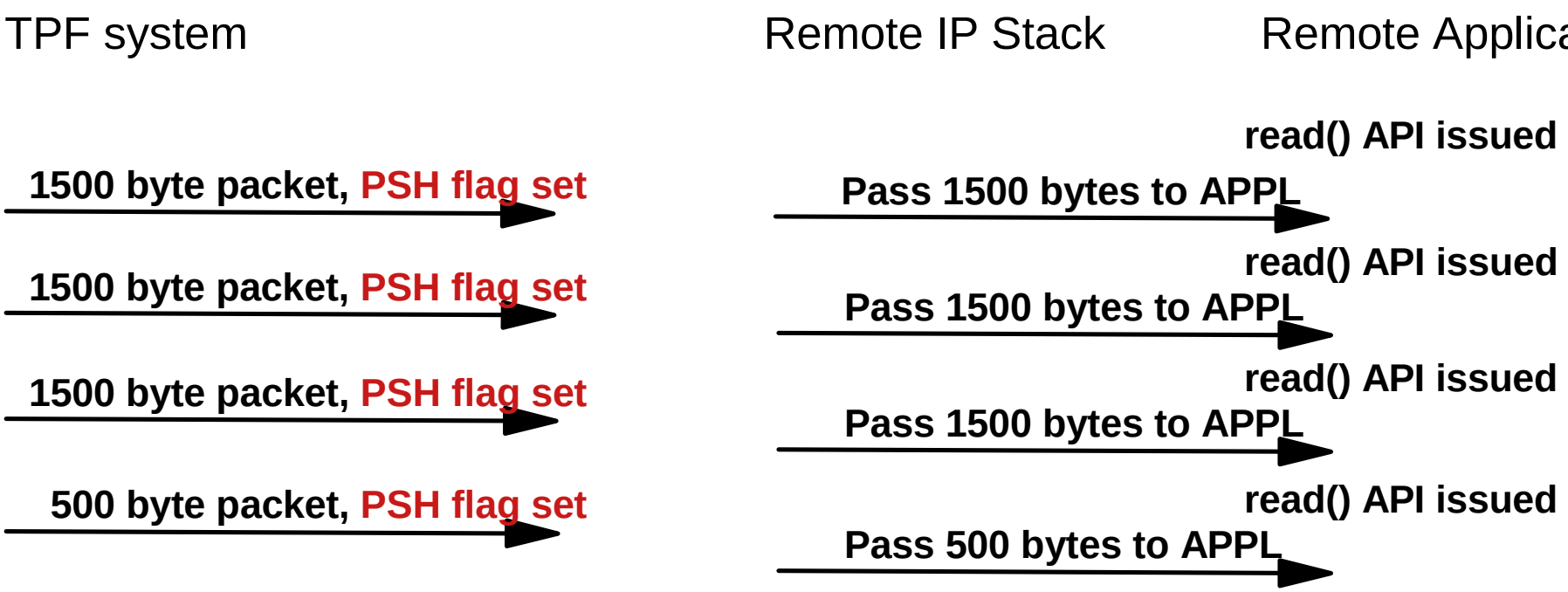
- TCP Architecture failed to standardize the use of the PSH flag
  - How and when to set the PSH flag is implementation dependent
  - How to process packets received with or without PSH set is also implementation dependent
  - There is no way for the application to control when to set the PSH flag

What is TPF's PSH Enhancement?

- Today, the PSH flag is set on every outbound packet from TPF
  - For many applications, this is optimal.
- Customer requirement for TPF application control of when the PSH flag is set
- New option on ioctl() API, TCP_PSH_LAST
  - Only sets the PSH flag on the last packet sent for a given send-type API.
  - Default is to set the PSH flag on every outbound packet

Example of PSH Flag Set For Every Packet

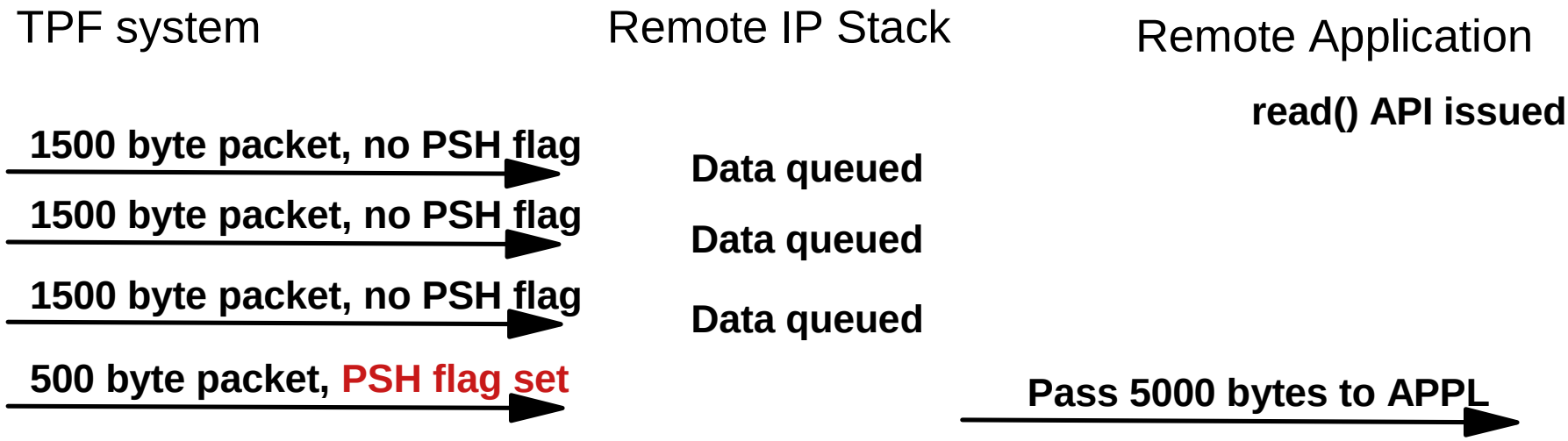
- Remote node passes each packet up to the application
- Let's assume the message sent is 5000 bytes long
  - One send API is equivalent to one message



- Application issues 4 read() API's to read a single message

Example of PSH Flag Set For The Last Packet

- The remote attempts to queue the data until PSH is received, otherwise it will be treated the same
- Let's assume the message sent is 5000 bytes long
  - One send API is equivalent to one message

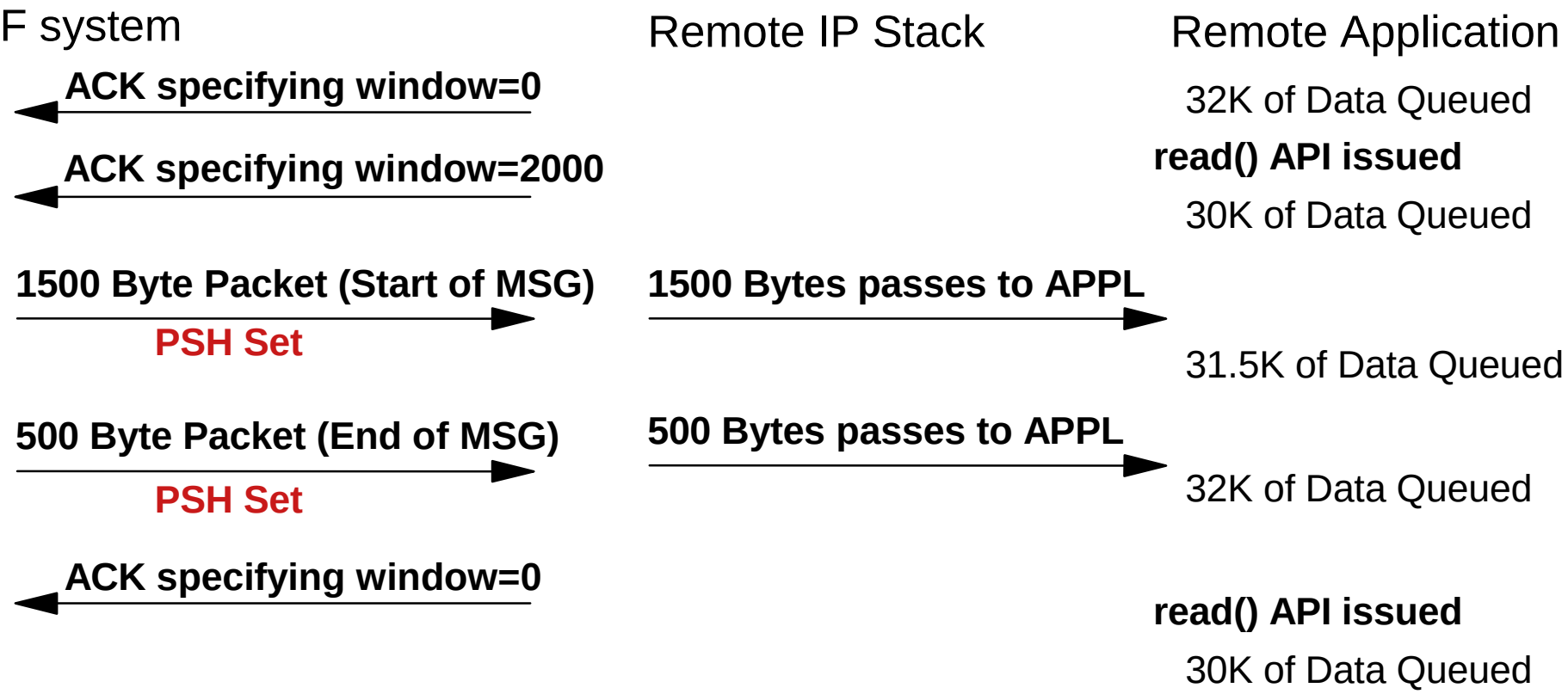


- By setting the PSH flag in only the last packet, the number of read() APIs issued is reduced to one for each message

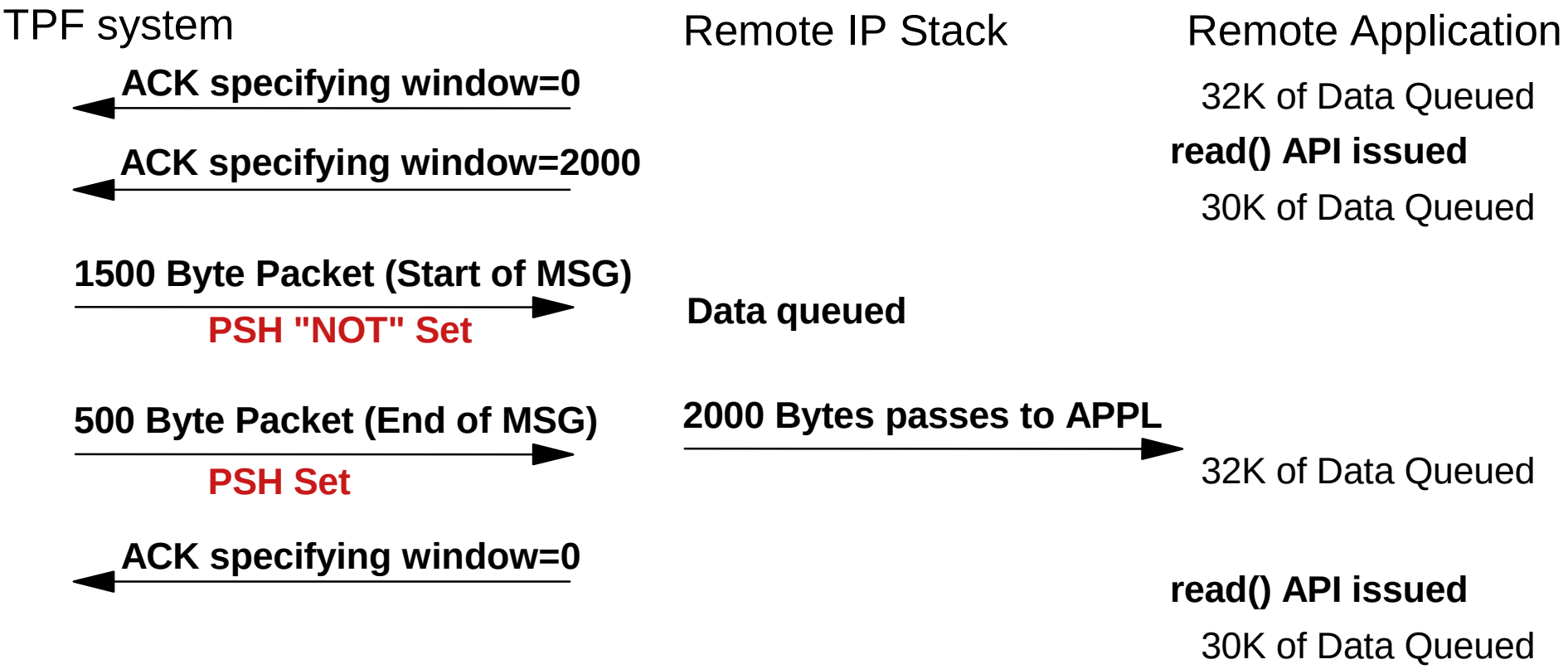
Case where PSH Flag Being Set has no Effect

- PSH flag being set can have no effect when data is arriving faster than the application can process it.
  - PSH flag is not a factor because data is always available to the application
- In this case the remote socket's receive buffer will be full until another message is read by the remote.
  - At which time, TPF will send the next message
- In the following example, TPF floods the remote with data
  - Fills up the remote socket's receive buffer.
  - TPF is blocked from sending until remote read's another message
  - Each message is 2000 bytes long
  - Remote using a 32K receive buffer
  - TPF local application issues sends for 2000 bytes
    - 1 message for each send (Spans two packets)

Example 1, when PSH does not matter



Example 2, when PSH does not matter

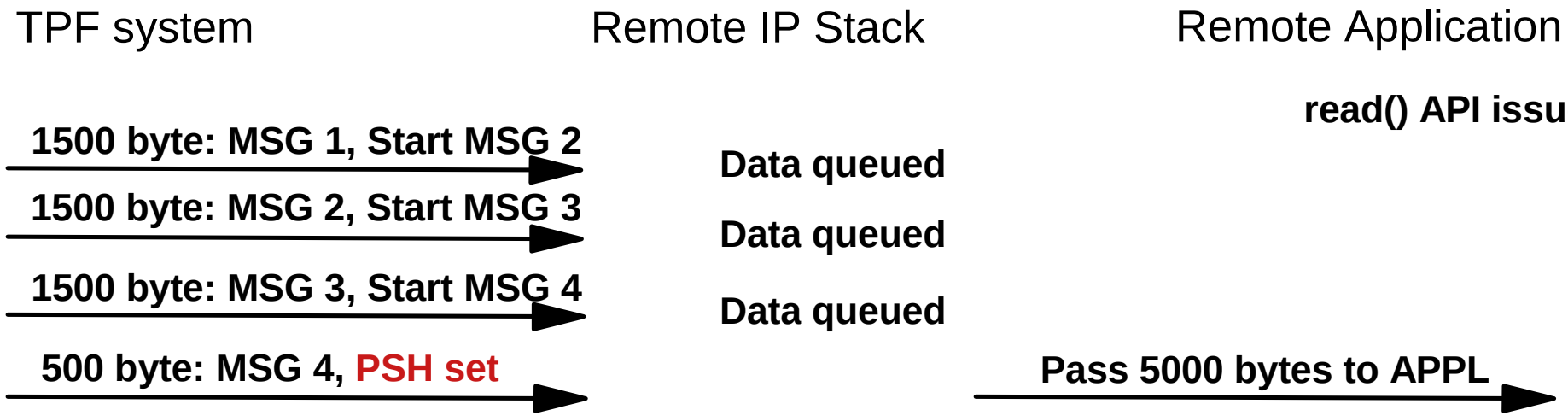


Why Don't I Always Use The PSH Enhancement?

- Some applications are better suited for having the data passed to them immediately (PSH flag set on every packet)
  - For example, the FTP Server
    - Want to pass the data received immediately to application so it can begin writing it out to file
- In some cases, applications issue sends with multiple messages contained in the data
  - Want the application to receive this data immediately
    - Won't have to wait for all messages to be received before processing the first one.

Example of Multiple Messages, PSH Last Set

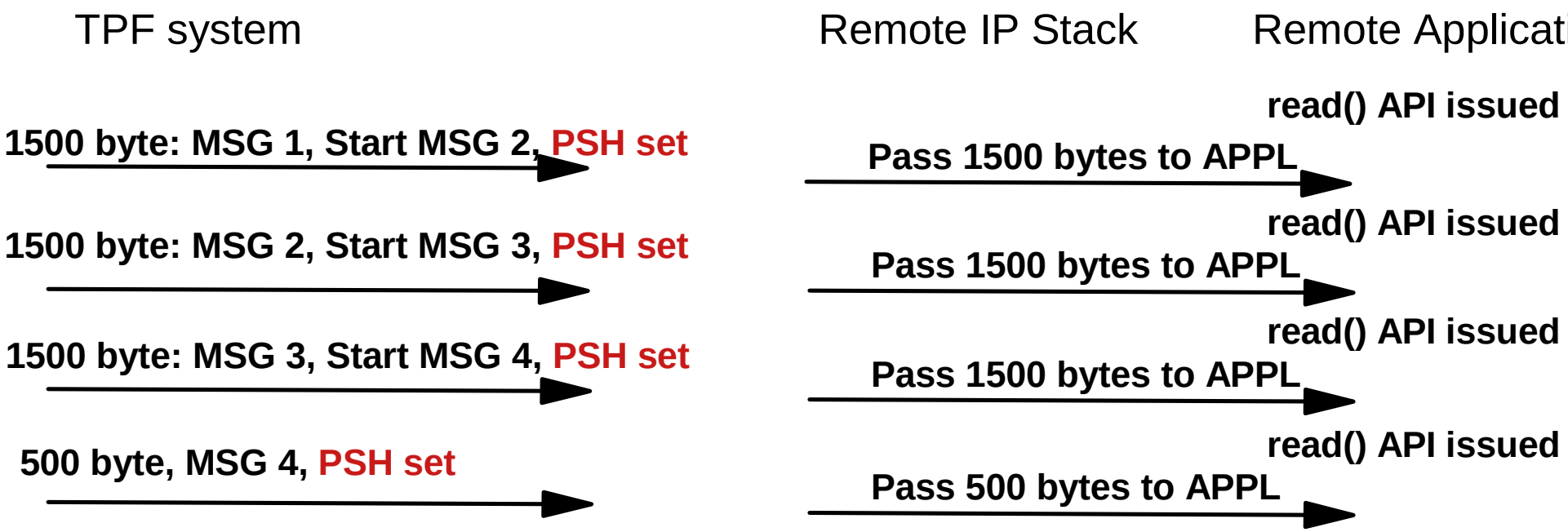
- For performance reasons, could have multiple messages sent on one send() API.
- In the example below, the TPF system issues a send for 5000 bytes
  - This send contains 4 separate messages for the application



- In this case, remote application has to wait for all four messages to be received before processing the first message

Example of Multiple Messages, PSH on Every Packet

- For performance reasons, could have multiple messages sent on one send() API.
- In the example below, the TPF system issues a send for 5000 bytes
  - This send contains 4 separate messages for the application



- In this case, remote application can begin processing message 1, while messages 2, 3, and 4 are being received

10 GbE OSA-Express Support PJ29930

10 GbE OSA-Express Support

- TPF currently supports two types of OSA-Express cards
  - Fast Ethernet OSA-Express
  - 1 GbE OSA-Express
- IBM has announced the support of a 10 gigabit OSA-Express card
  - Support to be delivered on the IBM z990 processor
- APAR PJ29930 allows TPF to exploit the 10 GbE OSA-Express card on the new processor

Recent TCP/IP APARs of Interest

Recent APARs of Interest

- PJ29273, PJ29411, PJ29424
  - Correct problems with APAR PJ28849
    - PJ28849 corrected discrepancies with MSS value for a given socket
  - Without fixes TCP connection requests can inadvertently be rejected and SNAP dumps can occur when UDP packets with zero data are received
- PJ29348
  - CTL-C can occur when TPF is cycling below CRAS state with many sockets with AORs pending.
    - When sockets are cleaned up, an ECB is created for each socket

Recent APARs of Interest

- PJ29450
  - With PJ28213 applied (TPF's packet filtering support), CTL-4 or CTL-1 system errors can occur out of TPF's socket sweeper support
    - When a TCP socket is cleaned up, the counter register for the number of sockets is corrupted.
- PJ29484
  - With PJ28195 applied (TPF's Network Services Database Support), CTL-1 dumps can occur out of CCTCP1
    - Occurs when searching for an application in the NSD, but the first attempt fails.

Recent APARs of Interest

- PJ29496
  - Performance Improvement to TPF's AOR timer chain processing
  - Problems occur when many sockets are active in the system issuing AOR
    - Only occurs when sockets define a timeout value for AOR's or when PJ28901 (Traffic Limiting) is applied.
  - Each time a message is received by TPF on a socket with AOR pending, the AOR timer chain must be scanned to remove the socket from the AOR timer chain
  - Chain can be very long for a serial search.
    - Enhanced to avoid serial searching of chains

Recent APARs of Interest

- PJ29770
  - With SNMP support applied (PJ27932), a possible CTL-1 can occur if an IGMP multicast message is received by TPF
- PJ29931
  - With the delayed ACK enhancement applied (PJ29638), a CTL-2 can occur when data out of order is received
- PJ29929 (HIPER APAR)
  - PJ29447 added code to account for the TCP shrinking window condition.
    - Will cause socket connections to get out of sync and eventually stall.

Questions?