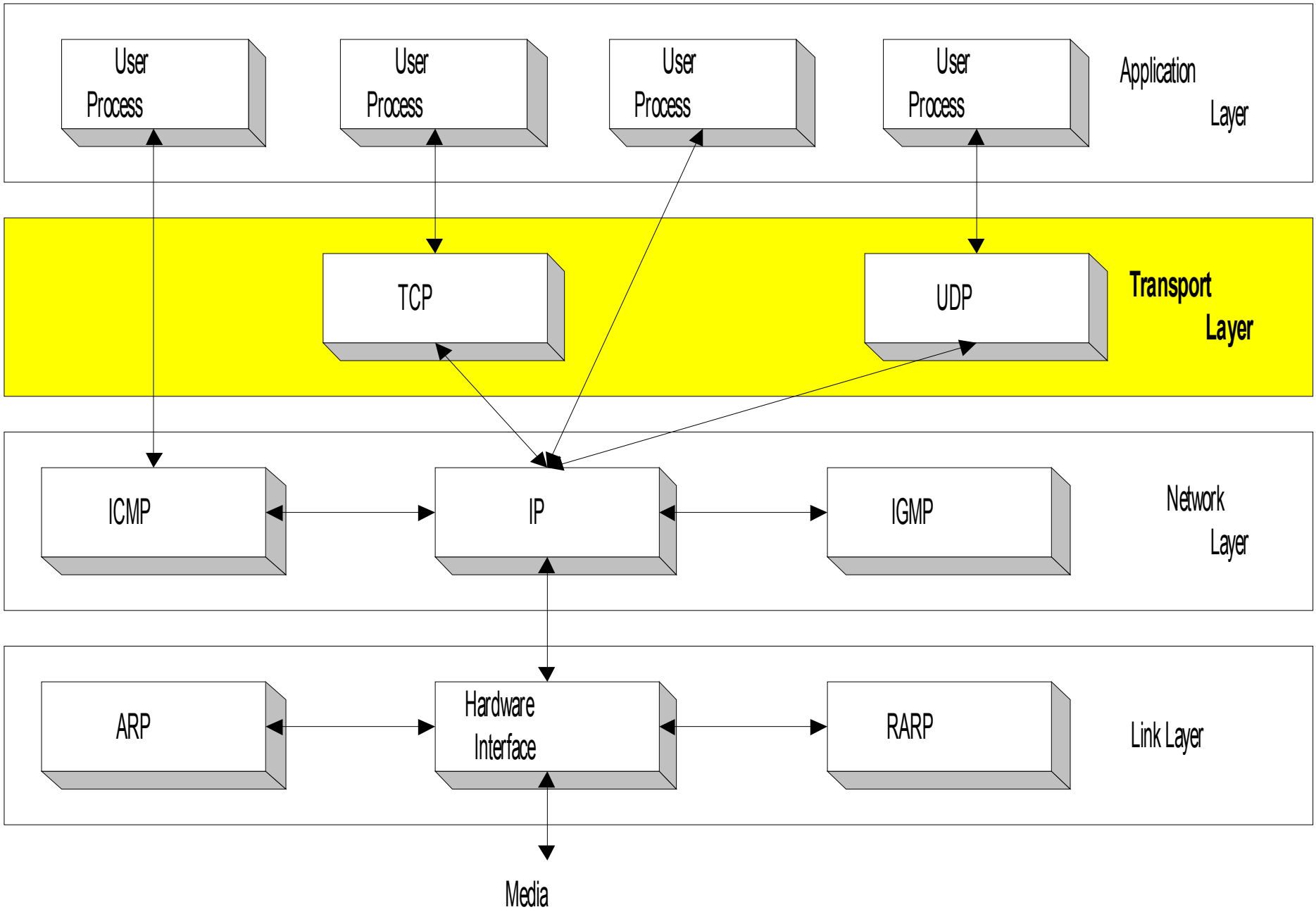
TCP-Verbindungen und Datenfluss

Jörn Stuphorn
stuphorn@rvs.uni-bielefeld.de

Stand der Veranstaltung

13. April 2005	Unix-Umgebung
20. April 2005	Unix-Umgebung
27. April 2005	Unix-Umgebung
4. Mai 2005	ARP, ICMP, ping
11. Mai 2005	IP-Adressen & Subnetzmasken
18. Mai 2005	Einführung in Bridging, Routing, ...
25. Mai 2005	IOS, Spanning-Tree
1. Juni 2005	IOS Befehle, Brigding, Routing
8. Juni 2005	Statisches Routing
15. Juni 2005	UDP-, MTU- und IP-Fragmentierung
22. Juni 2005	***TCP-Verbindungen und -Datenfluss***
29. Juni 2005	*DHCP und NTP*
6. Juli 2005	*NAT und Firewalls*
13. Juli 2005	*Verschlüsselung, Vertraulichkeit, Authentisierung*
20. Juli 2005	*Sichere Anwendungen*

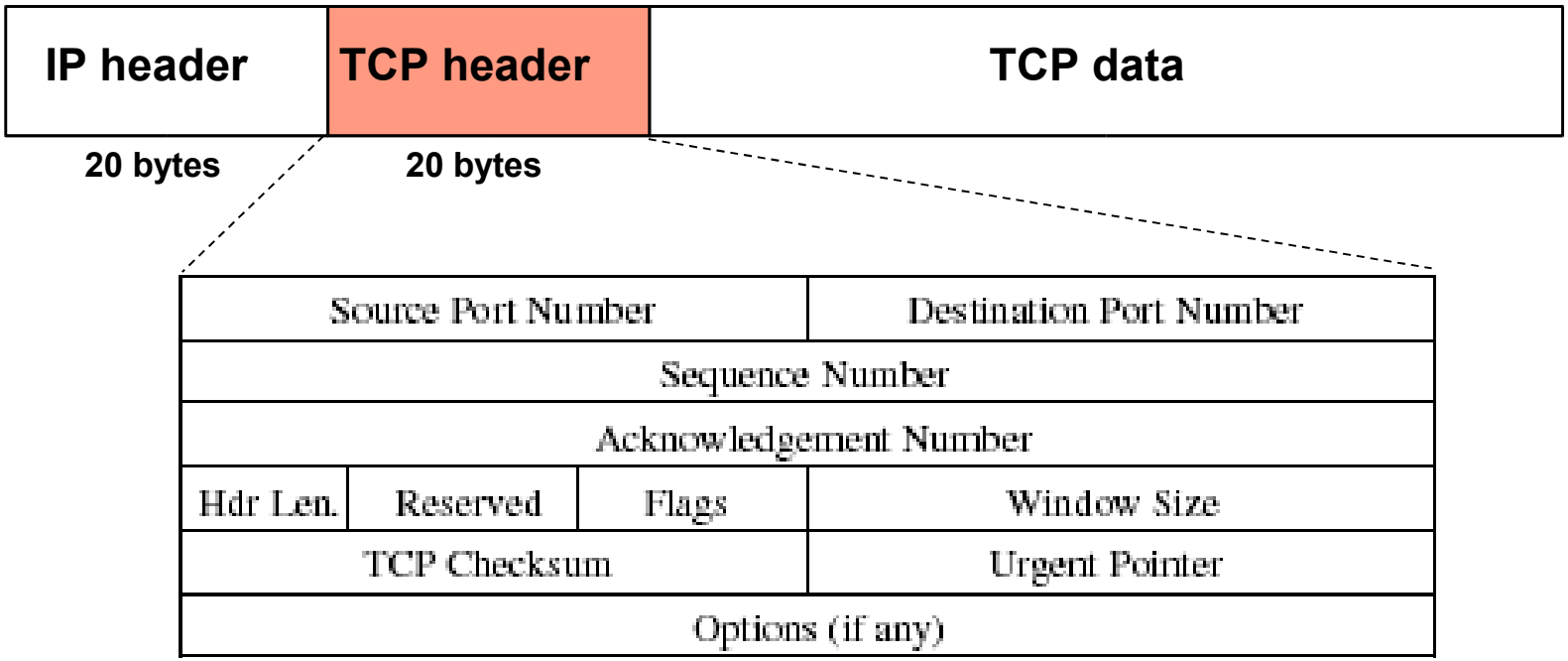
TCP/IP Stack


User Process
User Process
User Process
User Process
Application Layer
TCP
UDP
Transport Layer
ICMP
IP
IGMP
Network Layer
ARP
Hardware Interface
RARP
Link Layer
Media

TCP Überblick

- TCP arbeitet auf Transport-Layer Schicht
- Liefert:
  - verbindungsorientierte und
  - zuverlässige Dienst an Anwendungen
- Beispiele:
  - HTTP,
  - eMail,
  - FTP,
  - telnet
- Unterstützt nur Unicast
- Features:
  - Fehlerkontrolle
  - Flusskontrolle
  - Überlastungskontrolle

TCP Header



- Quell- und Ziel-Portnummer
- Sequenz- und Bestätigungsnummer
- Headerlänge
- Flags
- Checksumme über Header und Payload

Header Felder

- Sequenznummer
  - 32bit
  - bezeichnet die Position des erste Bytes des Segments im TCP Stream
- ACK Nummer
  - 32bit
  - die Sequenznummer, die der Host als nächstes empfangen möchte
- Windowgröße
  - 16bit, maximale Segmentgröße (in Byte), die der Empfänger akzeptiert
- Flags
  - URG: dringende Nachricht wird übertragen
  - ACK: Bestätigungsnummer ist gültig
  - PSH: Empfänger soll Informationen schnellst möglich an Anwendung weiterleiten
  - RST: Signal um TCP Verbindung zurückzusetzen
  - SYN: Signal zum Verbindungsaufbau
  - FIN: Signal zum Verbindungsabbau
- Urgent Pointer
  - 16bit, wenn URG-Flag gesetzt ist, zeigt Pointer auf letztes Byte der dringenden Nachricht in der TCP Payload

TCP Verbindung

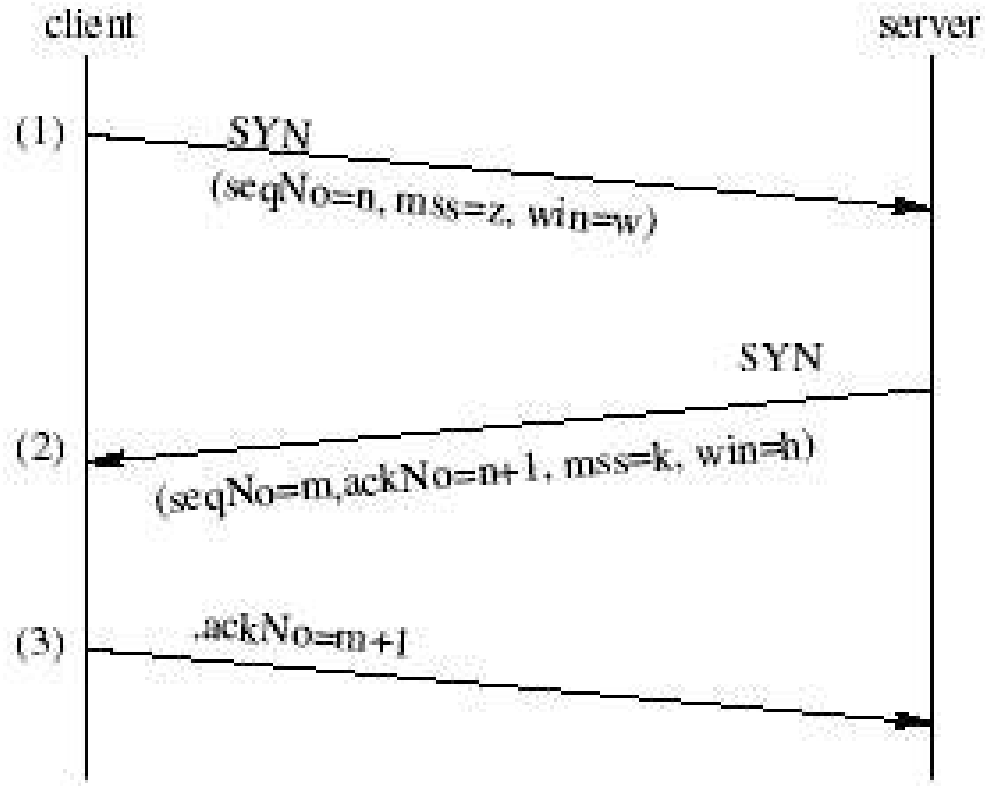
- Portnummern kennzeichnen sendende und empfangende Anwendungen
- Kombination aus IP Adresse und Portnummer heißt **Socket**
- TCP Verbindung wird durch zwei Sockets eindeutig identifiziert

Verbindungs-Management

- Verbindungsaufbau
  - Reservierung benötigter Resourcen
  - Aushandlung der Übertragungsparameter
    - maximale Segmentgröße
    - Empfangspuffer
    - Initiale Sequenznummer (ISN)
- Verbindungsabbau
  - TCP Verbindung ist Full Duplex
  - jeder Host muss seinen ausgehenden Datenfluss stoppen
  - anschließend wartet er auf verspätete Segmente
  - Wenn ein nicht behebbarer Fehler erkannt wurde kann jeder Host mit RST die TCP Verbindung abbauen
- TCP Timer

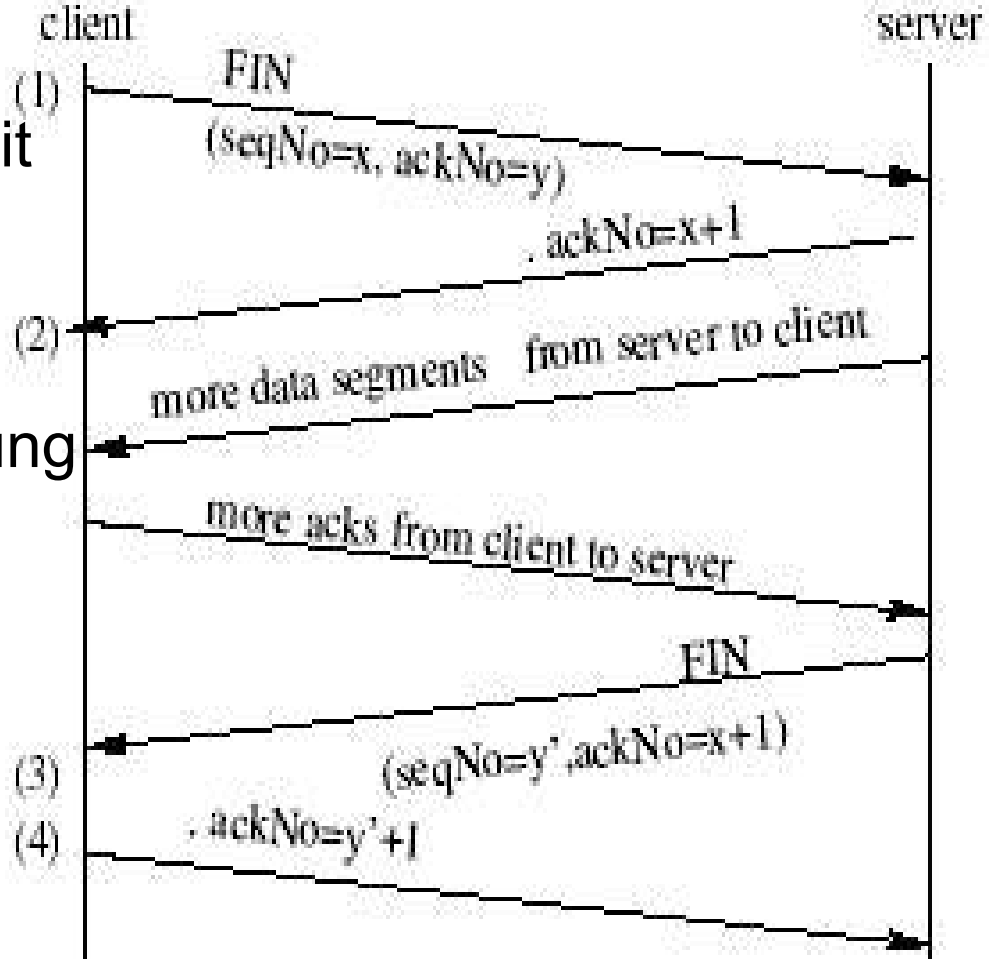
Verbindungaufbau

- 3-Wege Handshake
- Ein Host sendet ein Paket mit
  - ISN, n
  - leerer Payload
  - MSS
  - Größe des Empfangsfensters
  - gesetztem SYN Flag
- Gegenstelle antwortet mit
  - ACK = n+1
  - eigene ISN, m
  - MSS
  - Größe des Empfangsfensters
- Initierender Host sendet
  - ACK = m+1



Verbindungsabbau

- 4-Wege Handshake
- TCP Half-Close
  - Ein Host sendet TCP Paket mit gesetzem FIN Flag
  - Gegenstelle bestätigt das FIN-Segment
  - Der Datenfluss in Gegenrichtung ist noch aktiv
- Anschließend erfolgt Half-Close in Gegenrichtung



TCP Timer (1)

- Verbindungsaufbau Timer
  - maximale Periode, bevor TCP den Verbindungsaufbau aufgibt
- Retransmission Timer
  - Ein TCP Segment wurde nicht bestätigt
  - Timeout des Timers
  - TCP Segment wird erneut gesendet
- Delayed ACK Timer
  - in interaktivem Datenfluss verwendet
- Keepalive Timer
  - eine Gegenstelle ist inaktiv
  - Timeout des Timers
  - Host kontrolliert, ob die Gegenstelle noch existiert

TCP Timer (2)

- TCP Persist Timer
  - Bei schnellem Sender und langsamen Empfänger
  - Gegenstelle meldet als Fenstergröße NULL
  - Timeout des Timers
  - Sender überprüft die Fenstergröße des Empfängers erneut
  - Verwendet Exponential Backoff Algorithmus
- Maximum Segment Life Wait Timer
  - Periode, die eine TCP Verbindung aktiv bleibt, nachdem letztes ACK-Paket des 4-Wege Handshakes (Abbau) gesendet wurde
  - Verhindert, dass
    - Verspätete Segmente einer vorhergehenden Sitzung als
    - Teile einer aktuellen Sitzung mit gleichem Socket interpretiert werden

Übung 1

- Traffic mit Ethereal auf beiden Maschinen protokollieren
- yoda: **telnet** [remote-host] **time**

- Welche Nachrichten wurden zum Verbindungsaufbau generiert?
- Welche MSS Werte wurden bekannt gegeben?
- War DF Flag gesetzt?

Übung 2

- Traffic mit Ethereal auf beiden Maschinen protokollieren
- yoda: sende ein UDP Datagramm an windu
- **sock -u -n** [remote-host]:**8888**

- yoda: sende ein TCP Stream an windu
- **sock -n** [remote-host]:**8888**

- Was ist beim UDP passiert?
- Was ist beim TCP passiert?
- Wie gehen UDP und TCP mit nicht vorhandenen Servern um?

TCP Datenfluss

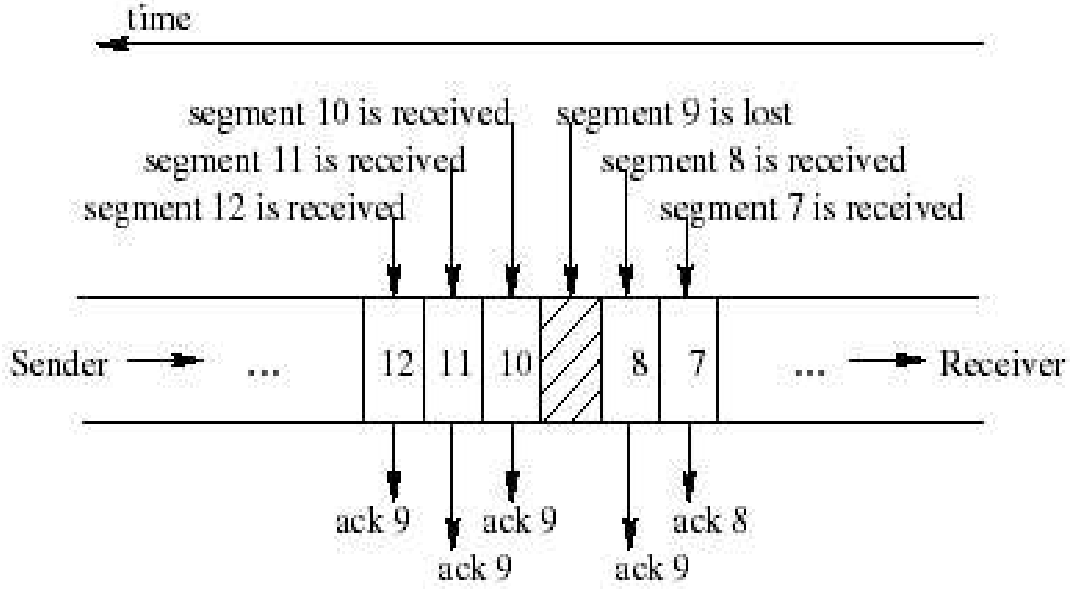
- TCP liefert eine Byte-Stream Verbindung an das Application Layer
- TCP Modul des Senders:
  - Empfang eines Streams von der Applikation
  - Bytes werden in Sendepuffer gelegt
  - Modul nimmt Daten aus Puffer und übergibt Segmente an IP Layer
- TCP Modul des Empfänger:
  - Empfangene Segmente werden in Puffer gelegt
    - Zwischenspeicherung
    - Reorganisierung empfangener Segmente
  - Erstellt Byte Stream aus Empfangspuffer
  - Stream wird an Anwendung gesendet

Fehlerbehandlung

- TCP Segmente können
  - verloren gehen
  - unsortiert empfangen werden
- TCP verwendet IP zur Übertragung
- IP: verbindungslos, nicht zuverlässig
- Fehlererkennung für Anwendungsdaten durch erneute Übertragung
  - fehlerhafter TCP Segmente und
  - verlorengegangener TCP Segmente

Fehlererkennung

- Jedes Datenbyte eindeutig über Sequenznummer identifiziert
- Positive Bestätigung über korrekt empfangene Bytes
- Fehlererkennung auf jeder TCP/IP Schicht mittels Header Prüfsummen
- Fehlerhafte Segmente verfallen
- Bei Verfall eines Segments:
  - ACK an Sender für 1. Byte des verfallenen Segments
  - Eine Lücke in den empfangenen Sequenznummern indiziert
    - Informationsverlust oder
    - Segmente nicht in richtiger Folge empfangen



Interaktiver Datenfluss

- Interaktive Anwendungen:
  - telnet
  - ssh
- TCP liefert interaktiven Datenfluss für interaktive Verbindungen
- Ziel: Reduzierung der Verzögerung die Benutzer erfährt
  - Benutzereingabe wird zuerst von Benutzer an Server gesendet
  - Server „echoes“ Eingabe zurück an Benutzer und überträgt ACK im Huckepack („piggybacking“)
  - User sendet ACK für Echo-Segment an Server und gibt Echo-Information auf Bildschirm aus.
- Extremfall: Ein Segment für jeden Tastendruck
- Effizienter:
  - Reduzierung benötigter Zahl kleiner Segmente
    - Delayed Acknowledgment
    - Nagle Algorithmus

Delayed ACK

- Delayed Acknowledgment Timer reagiert alle X ms (z.B. 50ms)
- TCP bestätigt Datensegmente erst beim nächsten „Tick“ des Timers
  - Wenn inzwischen weitere Daten übertragen werden
    - ACK im Huckpack-Verfahren
  - Sonst, senden eines ACK Segments
- Resultat: Ein ACK kann zwischen 0 und X ms verzögert werden.

Nagle Algorithmus

- Jede TCP Verbindung kann nur ein nicht bestätigtes Segment haben
- TCP sendet ein Byte und puffert alle weiteren Byte bis eine Bestätigung für des erste Byte empfangen wurde
- Alle gepufferten Byte werden in einem einzelnen Segment übertragen

- Effizienter als die Übertragung mehrerer Segmente je 1 Byte
- Größere Verzögerung für Benutzer

Übung 3

- Traffic mit Ethereal auf beiden Maschinen protokollieren
- yoda
  - **telnet** [remote-host]
  - Anmelden
  - **a) date**
  - b) Eine schnelle Zeichenfolge eingeben

- Sind delayed ACK erkennbar?
- Wenn ja, warum?
- Wenn nein, warum nicht?

- Wird der Nagle Algorithmus verwendet?

Bulk Data Flow

- TCP unterstützt Bulk Datenübertragung
- Große Menge von Bytes werden mit TCP übertragen
- Anwendungen: eMail, FTP, HTTP
- Stau (“congestion”) kann auftreten
  - wenn schneller Sender an langsamen Empfänger sendet
  - Pakete werden verworfen, wenn Puffer voll ist

  - Die Quelle will Senderate erhöhen um hohen Durchsatz zu erzielen
  - Die Senderate der Quelle sollte auf Maximalwert begrenzt werden, der ohne Stau oder Pufferüberlauf machbar ist.

Übung 4

- Traffic mit Ethereal auf beiden Maschinen protokollieren
- yoda: **sock -nl :7777**
- windu: **sock -n** [remote-host]:**7777**

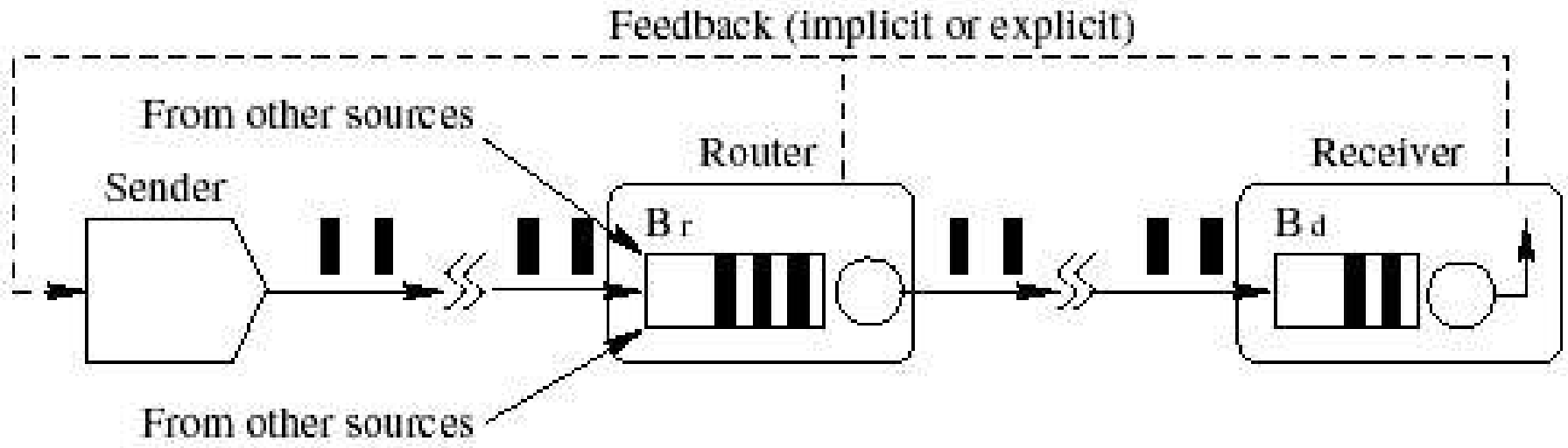
- Speicher die jeweiligen Protokolle

- Beschreibe TCP Operation in Hinsicht auf
  - Datensegmente
  - Bestätigung von Datensegmenten
- Gibt es Unterschiede in den Protokollen?
- Wie viele unterschiedliche TCP Flags treten auf?

Congestion & Flow Control

- Stau- und Flusskontrolle behandeln Überlastungsprobleme
- Ziel: Quelle soll sich an Pufferstatus von Routern und Receiver anpassen

- TCP verwendet langsamen Start und Überlastungsvermeidung
  - Reaktion auf Stauungen in Routern und
  - Vermeidung von Pufferüberläufen beim Empfänger

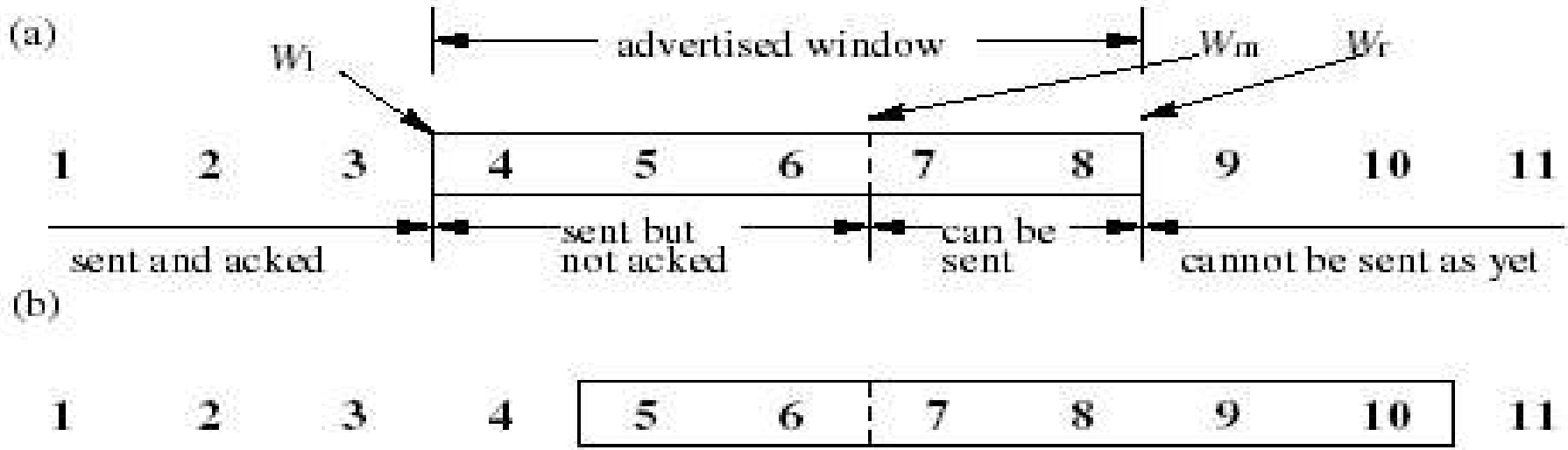


Sliding Window Flow Control

- Empfänger meldet maximal verarbeitbare Datenmenge
- Sender darf nicht mehr Daten als gemeldet übertragen

- Senderate wird bestimmt durch
  - Bekanntgegebenes Fenster
  - Geschwindigkeit, mit der Segmente bestätigt werden

- Überlastungen können immer noch auftreten



TCP Congestion Control

- Router-Puffer wird normalerweise geteilt
  - viele TCP Verbindungen
  - andere non-TCP Datenübertragungen
- TCP muss Senderate an Änderungen in übrigen Übertragungen anpassen
  - Schnellstmögliche Steigerung der Senderate neuer TCP Verbindugen
  - Verringerung der Steigerung, wenn Senderate größer als ein Schwellenwert ist
- Überlastungsmeldungen:
  - Sender kann aus Retransmission-Timeout auf Überlastung schließen
  - Empfänger meldet Überlastung mit mehrfacher Übertragung von Bestätigungen