



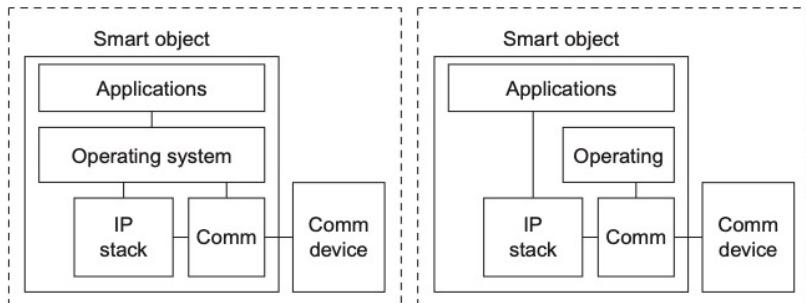
uIP – a Lightweight IP stack

The open source uIP IP stack was the first IP stack for smart objects.

To communicate using the IP, a device needs an IP stack. This is a software system that implements the IP protocols enabling IP communication. Every computer on the Internet runs an IP stack. They are part of all general purpose operating systems such as Microsoft Windows, Linux, and Mac OS. Smart objects are, however, severely memory-constrained and the IP stacks in general purpose computers require comparatively large amounts of memory. For example, the IP stack in Linux requires at least one megabyte of memory to maintain memory buffers for incoming and outgoing data. In contrast a smart object typically has only a few kilobytes of memory available.

Activities of the IP stack

- The IP stack takes care of the communication
- Applications can either use the IP stack through the operating system (left) or directly interface with the IP stack (right).
- The IP stack sends and receives packets from the communication device driver.



Ad alto livello, le attività dello stack IP sono semplici: invia e riceve pacchetti dal driver del dispositivo di comunicazione. Le applicazioni comunicano con lo stack IP tramite il sistema operativo o direttamente con lo stack IP. Quando un pacchetto arriva dal driver del dispositivo di comunicazione, lo stack IP analizza le intestazioni del pacchetto nel pacchetto, estrae tutti i dati dell'applicazione dal pacchetto e invia i dati all'applicazione. Quando un'applicazione desidera inviare dati, l'applicazione invia i suoi dati allo stack IP. Lo stack IP inserisce i dati dell'applicazione in un pacchetto, crea le intestazioni dei pacchetti necessarie e invia il pacchetto al driver del dispositivo di comunicazione per la trasmissione sul dispositivo di comunicazione. Oltre a rispondere alle richieste dirette dall'applicazione e ai pacchetti in entrata, lo stack IP si occupa anche dell'elaborazione periodica del protocollo come l'esecuzione di ritrasmissioni.

Lo stack uIP è stato originariamente progettato per essere utilizzato con e senza un sistema operativo, come mostrato in Figura. Oggi, molti sistemi operativi utilizzano lo stack uIP per la comunicazione IP. L'esempio più importante è il sistema operativo Contiki, che è anche l'attuale piattaforma di sviluppo per uIP. FreeRTOS offre una scelta dello stack uIP o del suo cugino più grande lwIP. TinyOS utilizza uIP per la comunicazione IPv4, ma recentemente ha incluso un'implementazione autonoma per IPv6.

IP for Embedded Systems

September 2001: first version of uIP

- World's smallest TCP/IP stack
- Fully RFC compliant
- IP, ICMP, UDP, TCP
- Meets the strict memory requirements of smart objects
- Open source license
- Bottom-up design
- uIP is the principal IP communication component of the Contiki operating system
- uIP is now used in a variety of systems and products:
 - oil pipeline monitoring systems, global container tracking systems, car engines, pico satellites

Lo stack uIP TCP / IP è un'implementazione dello stack IP appositamente progettata per soddisfare i rigidi requisiti di memoria di oggetti intelligenti e altri sistemi integrati in rete. La prima versione di uIP è stata rilasciata nel settembre 2001. È stata rilasciata con una licenza open source permissiva che consente al software di essere utilizzato liberamente in sistemi commerciali e non commerciali. Dalla sua prima versione, lo stack uIP ha visto un'importante adozione industriale e il software è ora utilizzato in sistemi e prodotti come sistemi di monitoraggio di oleodotti, sistemi globali di localizzazione dei container, motori di automobili e satelliti pico.

uIP è il principale componente di comunicazione IP del sistema operativo Contiki. Sebbene lo stack uIP possa essere utilizzato come pacchetto software autonomo e spesso venga utilizzato in questo modo, il suo continuo sviluppo viene eseguito all'interno di Contiki.

uIP ha requisiti di memoria molto bassi. Nella sua configurazione predefinita, richiede solo circa un kilobyte di RAM e pochi kilobyte di ROM. Ciò include i protocolli IP, ICMP, UDP e TCP. La dimensione del codice specifico dipende dal processore su cui viene utilizzato l'UIP. È possibile ridurre ulteriormente l'impronta RAM, ma a scapito della conformità standard. La configurazione più piccola richiede solo circa 100 byte di RAM, ma tale configurazione non è necessariamente conforme allo standard. Inoltre, la dimensione del footprint di memoria influisce sulla velocità di trasmissione dei dati ottenibile. Per molte applicazioni, tuttavia, un footprint di memoria ridotto è più importante di un throughput elevato.

Example: uIP-based Pico Satellite

- CubeSat: pico satellite construction kit
 - MSP430-based
 - 128 bytes of RAM for uIP



uIP features

- Very low memory requirements
 - In its default configuration, it requires only about one kilobyte of RAM and a few kilobytes of ROM. This includes the IP, ICMP, UDP, and TCP protocols
 - The specific code size depends on the processor on which the uIP is used
 - It is possible to reduce the RAM footprint further, but at the expense of standard compliance
 - The smallest configuration requires only about 100 bytes of RAM
- The size of the memory footprint affects the achievable data throughput. For many applications, however, a low memory footprint is more important than a high throughput

uIP ha requisiti di memoria molto bassi. Nella sua configurazione predefinita, richiede solo circa un kilobyte di RAM e pochi kilobyte di ROM. Ciò include i protocolli IP, ICMP, UDP e TCP. La dimensione del codice specifico dipende dal processore su cui viene utilizzato l'UIP. È possibile ridurre ulteriormente l'impronta RAM, ma a scapito della conformità standard. La configurazione più piccola richiede solo circa 100 byte di RAM, ma tale configurazione non è necessariamente conforme allo standard. Inoltre, la dimensione del footprint di memoria influisce sulla velocità di trasmissione dei dati ottenibile. Per molte applicazioni, tuttavia, un footprint di memoria ridotto è più importante di un throughput elevato

IP for Embedded Systems: History

- **Embedded systems**
 - Resource-constrained
 - Autonomous operation
 - Remote configuration, data collection, ...
- **1999: EMIT Alliance – remote communication**
 - EMbed the InterneT Alliance
 - EmWare, Atmel, Texas Instruments, Zilog, Microchip,
 - ...
 - ”IP too heavyweight
 - ✓ Non-IP based wired protocol
 - ✓ Gateways

2 marzo 1999 - Emware (Salt Lake City, Utah) ha annunciato che Motorola, l'Electronic Device Group di Mitsubishi Electronic America e National Semiconductor Corp. hanno aderito all'Embed The Internet (ETI) Alliance. Fondata da Emware, ETI Alliance è composta da 16 produttori di microcontrollori, fornitori di applicazioni aziendali e strumenti integrati. Questo gruppo progressista si impegna a promuovere la piattaforma EMIT (Embedded Micro Internetworking Technology) di Emware come standard per connettere dispositivi elettronici a 8 e 16 bit nel processo di gestione aziendale, fornendo agli sviluppatori standard end-to-end, soluzioni aperte per il networking di dispositivi embedded.

2000: lwIP (Lightweight TCP/IP stack)

- Open source lightweight IP stack
 - Top-down design
 - Similar to BSD, Linux, ...
 - 40 kB RAM, 20 kB ROM
 - IP, ICMP, UDP, TCP
 - Widely used
 - “Too large” ...

The benefit of the lwIP stack is the higher performance it achieves when compared to the uIP stack.

Lo stack uIP è stato sviluppato circa un anno dopo il rilascio dello stack lwIP (stack TCP / IP leggero). Lo stack lwIP è progettato per sistemi leggermente più grandi di uIP e richiede una maggiore quantità di memoria sia per il buffering che per l'archiviazione del codice eseguibile. Un'installazione lwIP richiede in genere circa 40 kB di RAM e 20 kB di ROM. Il vantaggio dello stack lwIP è rappresentato dalle prestazioni più elevate rispetto allo stack uIP.

Berkeley Software Distribution (BSD, a volte chiamato Berkeley Unix) è un derivato del sistema operativo Unix sviluppato e distribuito dal Computer Systems Research Group (CSRG) dell'Università della California, Berkeley, dal 1977 al 1995. Oggi il termine "BSD" è spesso usato in modo non specifico per fare riferimento a uno qualsiasi dei discendenti di BSD che insieme formano un ramo della famiglia di sistemi operativi simili a Unix.

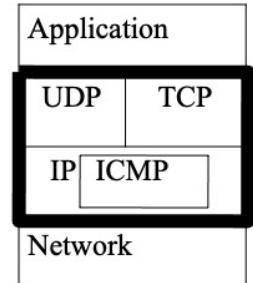
2001: uIP

- World's smallest TCP/IP stack
 - Open source
 - Bottom-up design
 - 4 kB ROM, 1 kB RAM
 - Fully RFC compliant
 - IP, ICMP, UDP, TCP

uIP ha requisiti di memoria molto bassi. Nella sua configurazione predefinita, richiede solo circa un kilobyte di RAM e pochi kilobyte di ROM. Ciò include i protocolli IP, ICMP, UDP e TCP. La dimensione del codice specifico dipende dal processore su cui viene utilizzato l'UIP. È possibile ridurre ulteriormente l'impronta RAM, ma a scapito della conformità standard. La configurazione più piccola richiede solo circa 100 byte di RAM, ma tale configurazione non è necessariamente conforme allo standard. Inoltre, la dimensione del footprint di memoria influisce sulla velocità di trasmissione dei dati ottenibile. Per molte applicazioni, tuttavia, un footprint di memoria ridotto è più importante di un throughput elevato.

Background: uIP – the world's smallest TCP/IP stack (2001)

- uIP – micro IP
- Open Source
- ~5k code, ~2k RAM
 - Smallest configuration ~3k code, ~128 bytes RAM
- RFC compliant
 - Order of magnitude smaller than previous stacks
- Bottom-up design
 - Single packet buffer
 - Event-driven API



Lo stack uIP implementa i protocolli di livello di rete e trasporto della famiglia di protocolli IP: IP, ICMP, UDP e TCP. È stato il primo stack IP per sistemi embedded a implementare completamente il protocollo TCP in modo da renderlo completamente compatibile con gli standard.

2002: IP for Embedded is a Fact

- Non-IP solutions went away
 - Highly optimized custom protocols
 - IP took over
- EMIT Alliance is no more
 - EmWare moved to uIP

The Most Importante Lesson Learned

- Ability to communicate more important than performance

2004: IP-based Sensor Networks

- Lightweight IP stack (uIP)
- Header compression
- Low-power MAC layer
- Sensor data
 - UDP
- Configuration
 - TCP
- 2006: IETF 6lowpan



Why IP for Sensor Networks?

- Scalable
- Versatile
- Manageable
- Open and flexible standard
- Widely deployed
- Lightweight
 - As shown by uIP, uIPv6
- Low-power
 - IETF 6lowpan WG
 - Power-saving MAC protocols

September 2008: IP for Smart Objects Alliance



October 2008: uIPv6, first certified IPv6 stack, 6lowPAN



Le prime versioni dello stack uIP presentavano solo la comunicazione IPv4, ma nel 2008 Cisco Systems ha esteso l'UIP con funzionalità IPv6. È stato sviluppato da Julien Abeillé e Mathilde Durvy. Lo stack uIPv6 è stato il primo stack a soddisfare tutti i requisiti IPv6, che gli ha consentito di utilizzare il logo Ready IPv6, come mostrato in Figura (uIP è stato il primo stack IPv6 per oggetti smart a essere certificato nel programma Ready IPv6. È certificato come IPv6 Ready, può utilizzare il logo IPv6 Ready).

Le estensioni uIPv6 sono state anche rilasciate come software open source e aggiunte al sistema operativo Contiki, rendendolo ampiamente disponibile.

uIP utilizza tre metodi per ridurre le dimensioni del codice e l'utilizzo della memoria: un'interfaccia di programmazione basata sugli eventi, uno schema di gestione del buffer intenzionalmente semplice e un'implementazione TCP efficiente in termini di memoria. Ritorniamo a questi meccanismi dopo aver esaminato il modo in cui uIP elabora i pacchetti in entrata e in uscita.

November 2008: 30th best invention of 2008

TIME IN PARTNERSHIP WITH **CNN** 49 Best Inventions

30. The Internet Of Things

In September, a group of high-tech companies that includes Cisco and Sun formed the IP for Smart Objects Alliance. Simply put, the organization intends to create a new kind of network that will allow sensor-enabled physical objects — appliances in your home, products in a factory, cars in a city — to talk to one another, the same way people communicate over the Internet.

ARTICLE TOOLS

- Print
- Email
- Sphere
- AddThis
- RSS
- Yahoo! Buzz

ILLUSTRATION FOR TIME BY CHRISTOPH NIEMANN

Top Stories

- Readyng Bu
- More Allegat
- Governor Ca
- Why Nicarg
- Where the R
- Europe's Hop
- Being Dashed

IwIP and uIP today

- Very well-known in the embedded community
- Used in products from companies
- Covered in several books on embedded networking
- Porting uIP in professional magazines
- Recommended by leading professionals
- Competence specifically required in job postings

Molto noto nella comunità embedded

Utilizzato in prodotti commerciali

Trattato in diversi libri sulle reti embedded

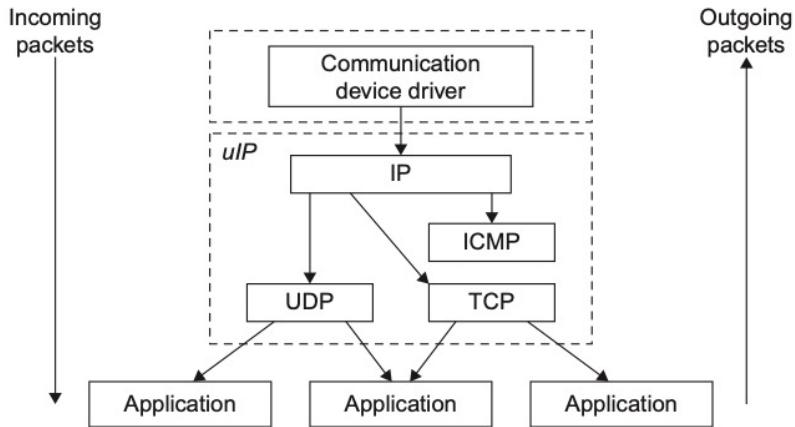
uIP su riviste professionali

Consigliato dai migliori professionisti

Competenza specificatamente richiesta negli annunci di lavoro

Principles of Operation

The principle of operation for uIP



- Incoming packets are passed to uIP from the communication device driver. After uIP has finished processing the data, if any, the packet is passed to the corresponding application
- Outgoing data pass through uIP, which adds protocol headers, before the packet is passed to the communication device driver for transmission

Il principio di funzionamento per lo stack uIP è semplice, come mostrato in Figura. Lo stack uIP fa tre cose: elabora i pacchetti che arrivano dal driver del dispositivo di comunicazione, elabora le richieste dall'applicazione ed esegue l'elaborazione periodica. Il modulo di inoltro uIP è responsabile dell'inoltro del traffico ad altri nodi. Il modulo di inoltro interroga un modulo di protocollo di routing per scoprire dove devono essere inoltrati i pacchetti.

L'elaborazione dell'ingresso inizia quando il driver del dispositivo di comunicazione ha ricevuto un pacchetto. Il driver chiama la funzione di elaborazione dell'input di uIP, che elabora le intestazioni del pacchetto in entrata, determina se il pacchetto contiene dati dell'applicazione e, in tal caso, passa i dati all'applicazione. L'applicazione può produrre una risposta ai dati in arrivo, che viene quindi gestita dalla parte di elaborazione dell'output di uIP.

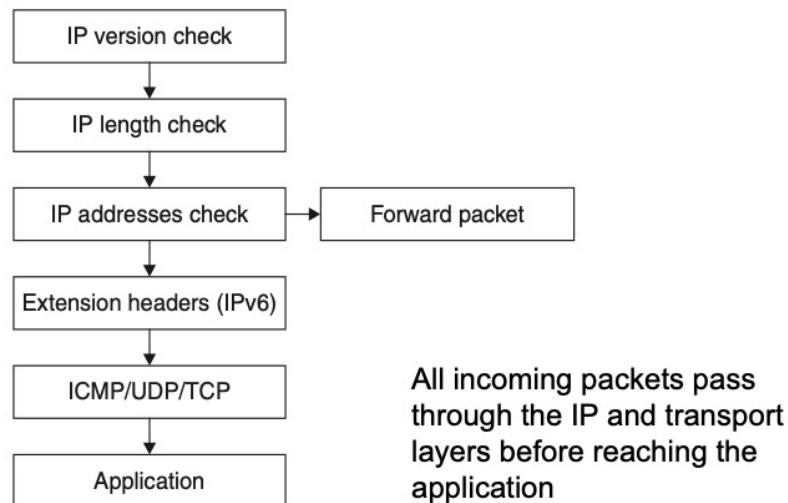
L'elaborazione dell'output è semplice. L'elaborazione dell'output si verifica dopo che l'applicazione è stata chiamata da uIP e solo quando l'applicazione ha prodotto dati da inviare allo stack uIP. Il codice di elaborazione dell'output aggiunge le intestazioni del protocollo al pacchetto che deve essere inviato e passa il pacchetto al dispositivo di comunicazione per la trasmissione.

L'elaborazione periodica viene eseguita per eseguire azioni basate sul timer come le ritrasmissioni. I meccanismi di elaborazione periodica in uIP sono intenzionalmente semplici. La funzione di elaborazione periodica uIP viene invocata regolarmente per verificare l'eventuale necessità di ritrasmissioni. In tal caso, produce il pacchetto da ritrasmettere e lo invia al driver del dispositivo di comunicazione che lo invia.

L'inoltro e il routing vengono eseguiti separatamente. L'inoltro è il processo di reinvio di un pacchetto ricevuto a un vicino, mentre l'instradamento è il processo per determinare a quali pacchetti dei vicini devono essere inoltrati. Il modulo di inoltro uIP mantiene una tabella di destinazioni e indirizzi del vicino hop successivo. I protocolli di routing, che in genere sono

implementati su UDP o TCP, popolano la tabella di inoltro in base ai dati ricevuti dal protocollo di routing.

Input processing in uIP



Quando il driver del dispositivo di comunicazione riceve un pacchetto dalla rete, chiama la funzione di elaborazione dell'input uIP per consegnare il pacchetto a uIP. Il codice di elaborazione dell'input uIP analizza le intestazioni dei pacchetti e determina se l'applicazione deve essere chiamata. In tal caso, uIP fornisce i dati all'applicazione.

L'elaborazione dell'input uIP inizia nell'intestazione IP. Il flusso della funzionalità di elaborazione dell'input è mostrato nella Figura. Le intestazioni dei pacchetti vengono analizzate dall'alto verso il basso, a partire dall'intestazione IP. Il codice di elaborazione dell'input uIP controlla innanzitutto il primo byte dell'intestazione IP per assicurarsi che il pacchetto in entrata sia un pacchetto IP e che la versione del protocollo IP corrisponda a quella che può essere gestita da uIP. uIP può attualmente gestire sia i dati IPv4 che IPv6, ma può gestirne solo uno alla volta.

Dopo aver verificato che il pacchetto abbia l'intestazione della versione IP corretta, uIP verifica la validità dell'intestazione IP. Verifica la lunghezza riportata nell'intestazione IP con la lunghezza del pacchetto ricevuto dal livello sottostante. Se la lunghezza nell'intestazione IP è più lunga del pacchetto nel buffer, il pacchetto viene considerato non valido e viene scartato. Se la lunghezza del pacchetto nel buffer è più lunga di quanto riportato nell'intestazione IP, si presume che il pacchetto sia ben formato, ma con dati di riempimento alla fine e uIP continua comunque a elaborare il pacchetto. Per IPv4, viene controllato il flag della frammentazione nell'intestazione IPv4. Se il pacchetto è un frammento IP, viene copiato nel buffer di deframmentazione. Se il buffer di frammenti contiene un pacchetto IP completo a causa del frammento in entrata, il pacchetto IP riassemblato viene visto come pacchetto IP in entrata e l'elaborazione dell'input continua. Per IPv6, il riassemblaggio del frammento IP viene eseguito come parte dell'elaborazione dell'intestazione dell'estensione.

Successivamente, vengono ispezionati gli indirizzi di origine e destinazione del pacchetto. I pacchetti con un indirizzo di origine IP non valido vengono eliminati. Esempi sono i pacchetti in cui l'indirizzo di origine è un indirizzo broadcast o multicast. I pacchetti con un indirizzo IP di destinazione che non corrisponde a nessuno degli indirizzi IP del nodo vengono scartati o consegnati al modulo di inoltro dei pacchetti uIP. Il modulo di inoltro dei pacchetti uIP può scegliere di inoltrare i pacchetti a un vicino di hop successivo, a seconda dell'indirizzo IP di destinazione.

I pacchetti con un indirizzo IP di destinazione che corrisponde a uno degli indirizzi IP del nodo vengono elaborati ulteriormente. Qui, il codice di elaborazione differisce tra IPv4 e IPv6. Per IPv4, il checksum dell'intestazione IP viene calcolato per assicurarsi che corrisponda. Per IPv6, non viene definito alcun checksum dell'intestazione. Per i pacchetti IPv6, uIP controlla il pacchetto per l'esistenza di eventuali header estesi e le elabora. Se uIP ha rilevato errori nelle intestazioni dell'estensione IPv6, viene generato un messaggio di errore ICMPv6 che viene inviato al mittente.

Quando uIP ha verificato che l'intestazione IP è corretta, che la lunghezza del pacchetto è corretta e che l'indirizzo di destinazione corrisponde all'indirizzo del nodo, il pacchetto viene assegnato al protocollo del livello di trasporto corretto. uIP supporta tre di questi protocolli: ICMP, UDP e TCP. ICMP, a rigor di termini, non è un protocollo di livello di trasporto ma è implementato come tale in uIP.

Il pacchetto viene elaborato in modo diverso a seconda del protocollo di livello superiore. TCP è il più complesso dei protocolli implementati da uIP. L'elaborazione UDP è molto semplice. L'elaborazione ICMP è molto semplice per IPv4 ma leggermente più complessa per IPv6.

ICMP Input Processing

- For IPv4, the ICMP processing consists of a single piece of functionality — to respond to incoming ICMP echo messages by sending an ICMP echo reply
 - ICMP echo messages are sent by the “ping” program present in most general purpose operating systems. The ping utility sends ICMP echo messages to check if a particular IP node is alive and for measuring the round-trip time to the node
- ICMP processing for IPv6 is more complex than IPv4 because ICMP has a more significant role in IPv6 than in IPv4
 - In addition to the echo functionality, ICMP in IPv6 is used for neighbor discovery (ND), router discovery, duplicate address detection, and other mechanisms. uIP supports ICMP neighbor solicitation messages, neighbor advertisement messages, router solicitation messages, and router advertisement messages

Per IPv4, l'elaborazione ICMP consiste in una singola funzionalità: rispondere ai messaggi di eco ICMP in arrivo. Quando un nodo riceve un messaggio di eco ICMP, risponde inviando una risposta di eco ICMP. Il messaggio di risposta contiene una copia dei dati nel messaggio di eco ICMP. I messaggi di eco ICMP vengono inviati dal programma "ping" presente nella maggior parte dei sistemi operativi di uso generale. L'utilità ping invia messaggi di eco ICMP per verificare se un particolare nodo IP è attivo e per misurare il tempo di andata e ritorno per il nodo. Quando il programma ping riceve una risposta echo ICMP che corrisponde al messaggio echo ICMP precedentemente inviato, il programma stampa il tempo di andata e ritorno sullo schermo.

L'elaborazione ICMP per IPv6 è più complessa di IPv4 poiché ICMP ha un ruolo più significativo in IPv6 rispetto a IPv4. Oltre alla funzionalità eco, ICMP in IPv6 viene utilizzato per il rilevamento dei vicini (ND), il rilevamento dei router, il rilevamento di indirizzi duplicati e altri meccanismi. uIP supporta i messaggi di sollecitazione del vicino ICMP, i messaggi di pubblicità del vicino, i messaggi di sollecitazione del router e i messaggi di pubblicità del router. In IPv6, i messaggi di sollecito vicini ICMP vengono utilizzati anche per eseguire il rilevamento di indirizzi duplicati, che è supportato da uIP.

UDP Input Processing

- UDP input processing in uIP is simple
- The input processing code first recalculates the UDP checksum to make sure that the packet is valid
- It uses the UDP port numbers in the packet to find the application to which the packet data should be delivered
- uIP maintains a list of applications and what UDP port numbers they use
- An application may specify the remote end point by providing the IP address and UDP port number of the remote peer, or it could leave them blank
 - Applications with blank remote IP address and UDP port numbers receive all packets that arrive for the application's UDP port
 - Otherwise, only packets from the specific IP address and UDP port

L'elaborazione dell'input UDP in uIP è semplice. Il codice di elaborazione dell'input prima ricalcola il checksum UDP per assicurarsi che il pacchetto sia valido. Quindi, utilizza i numeri di porta UDP nel pacchetto per trovare l'applicazione a cui consegnare i dati del pacchetto. uIP mantiene un elenco di applicazioni e quali numeri di porta UDP usano. Un'applicazione può specificare l'endpoint remoto fornendo l'indirizzo IP e il numero di porta UDP del peer remoto, oppure potrebbe lasciarli vuoti. Le applicazioni con indirizzo IP remoto vuoto e numeri di porta UDP ricevono tutti i pacchetti che arrivano per la porta UDP dell'applicazione. In caso contrario, vengono inviati all'applicazione solo pacchetti dall'indirizzo IP e dal numero di porta UDP specifici.

TCP Input Processing

- The most complex protocol implementation in uIP is TCP
- TCP processing begins by validating the TCP checksum
 - The TCP checksum is computed over the TCP header, the application data, and parts of the IP header
- After validating the checksum, the TCP port numbers of the packet are checked against the list of active TCP connections
 - If the packet does not match any of the current connections, uIP checks the list of listening TCP ports, but only if the incoming packet has the TCP SYN flag set
 - If the packet is not for an active connection or is not a TCP SYN packet for a listening port, uIP sends a TCP RST packet to the IP address that sent the packet
 - If the incoming packet was a TCP SYN packet for a listening connection, uIP creates a new entry in the table of active TCP connections and fills in the correct TCP sequence number from the incoming TCP SYN packet
 - If the incoming packet was destined for an active connection, uIP first ensures that the TCP sequence number of the incoming segment is what uIP expects

L'implementazione del protocollo più complessa in uIP è TCP. Tuttavia, l'implementazione è significativamente più semplice rispetto alle implementazioni TCP in altri stack IP. Ciò si verifica perché in uIP TCP esegue solo i meccanismi necessari per la conformità agli standard e l'interoperabilità da host a host. Poiché uIP è ottimizzato per un'occupazione di memoria ridotta e non per prestazioni elevate, l'implementazione TCP uIP si astiene dall'implementare una serie di meccanismi di ottimizzazione delle prestazioni presenti in altri stack IP.

L'elaborazione TCP inizia convalidando il checksum TCP. Il checksum TCP viene calcolato sull'intestazione TCP, i dati dell'applicazione e parti dell'intestazione IP.

Dopo aver convalidato il checksum, i numeri di porta TCP del pacchetto vengono verificati rispetto all'elenco delle connessioni TCP attive. Se il pacchetto non corrisponde a nessuna delle connessioni correnti, uIP controlla l'elenco delle porte TCP in ascolto, ma solo se il pacchetto in arrivo ha impostato il flag TCP SYN. Se il pacchetto non è per una connessione attiva o non è un pacchetto TCP SYN per una porta di ascolto, uIP invia un pacchetto TCP RST all'indirizzo IP che ha inviato il pacchetto.

Se il pacchetto in arrivo era un pacchetto TCP SYN per una connessione in ascolto, uIP crea una nuova voce nella tabella delle connessioni TCP attive e inserisce il numero di sequenza TCP corretto dal pacchetto TCP SYN in entrata. Se il pacchetto TCP SYN contiene un'opzione di dimensione massima del segmento TCP (MSS), uIP ricorda l'MSS che può inviare tramite questa connessione. Successivamente, uIP produce un pacchetto TCP SYN con il set di flag ACK e lo rimanda al peer remoto.

Se il pacchetto in entrata era destinato a una connessione attiva, uIP garantisce innanzitutto che il numero di sequenza TCP del segmento in entrata sia quello che si aspetta uIP. Se il numero di sequenza è superiore al previsto, indica che un pacchetto è stato perso. In tal caso, uIP scarta il pacchetto, sapendo che verrà ritrasmesso qualche tempo dopo. Sebbene sia possibile per uIP bufferizzare il pacchetto in arrivo in modo che sia immediatamente

disponibile una volta che il pacchetto mancante è stato ritrasmesso, ciò richiederebbe una memoria buffer non disponibile nei sistemi con limiti di memoria per i quali è progettato uIP.

TCP Input Processing (cont.)

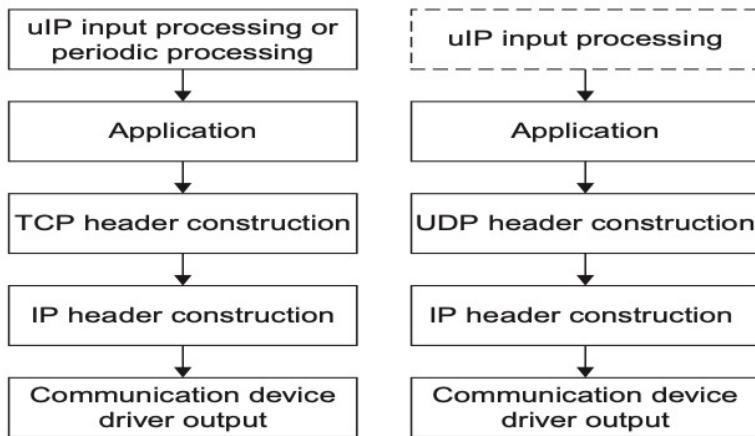
- After verifying the TCP sequence number, uIP performs a round-trip time (RTT) estimation mechanism
 - The purpose of the RTT estimation is for uIP to have a suitable value for the retransmission timer
 - uIP performs RTT estimation by keeping a counter for each TCP connection. The counter is incremented as part of the periodic uIP processing. When a TCP packet arrives, uIP uses the value of the counter as an estimate for the RTT of the packet
 - An averaging filter is used to provide a smoothed RTT estimate for the TCP connection
- Next, uIP takes different actions depending on the state of the TCP connection
 - If the TCP connection is established, the application is invoked and passes the application data present in the incoming TCP packet. The application consumes the data and may produce a reply and uIP sends it back to the remote peer
 - If the TCP connection for which the incoming packet is destined is in a state where it is about to be opened or closed, the TCP connection switches states as described in the TCP specification
 - When a TCP connection is opened or closed, uIP informs the application of the event by calling it. Depending on the state of the TCP connection, the application may choose to send data in response to the event. The packet then makes its way through the output processing code of uIP.

Dopo aver verificato il numero di sequenza TCP, uIP esegue un meccanismo di stima del tempo di andata e ritorno (RTT). Lo scopo della stima RTT è che uIP abbia un valore adatto per il timer di ritrasmissione. Per una connessione TCP con un RTT lungo, il timer di ritrasmissione deve essere impostato su un valore elevato e, al contrario, una connessione TCP con un RTT breve richiede un valore basso sul timer di ritrasmissione. uIP esegue la stima RTT mantenendo un contatore per ogni connessione TCP. Il contatore viene incrementato come parte dell'elaborazione periodica del UIP. Quando arriva un pacchetto TCP, uIP utilizza il valore del contatore come stima per l'RTT del pacchetto. Un filtro con calcolo della media viene utilizzato per fornire una stima RTT semplificata per la connessione TCP.

Successivamente, uIP esegue diverse azioni a seconda dello stato della connessione TCP. Se viene stabilita la connessione TCP, l'applicazione viene invocata e passa i dati dell'applicazione presente nel pacchetto TCP in entrata. L'applicazione consuma i dati e può produrre una risposta e uIP li restituisce al peer remoto. Se la connessione TCP a cui è destinato il pacchetto in entrata si trova in uno stato in cui sta per essere aperta o chiusa, la connessione TCP commuta gli stati come descritto nella specifica TCP.

Quando una connessione TCP viene aperta o chiusa, uIP informa l'applicazione dell'evento chiamandola. A seconda dello stato della connessione TCP, l'applicazione può scegliere di inviare dati in risposta all'evento. Il pacchetto quindi si fa strada attraverso il codice di elaborazione dell'output di uIP.

Output Processing



- For TCP connections (left), all output processing starts with uIP calling the application
- For UDP connections (right), applications may send UDP data directly but can also send data when called from the uIP input processing code

L'elaborazione dell'output in uIP è più semplice dell'elaborazione dell'input. L'elaborazione dell'output inizia quando uIP chiama l'applicazione. Quando viene chiamato da uIP, l'applicazione può scegliere di produrre un pacchetto. uIP aggiunge le intestazioni di pacchetto necessarie e passa il pacchetto, con le intestazioni, al driver del dispositivo di comunicazione per la trasmissione. La struttura dell'elaborazione dell'output uIP è mostrata in Figura.

Per le connessioni TCP, un'applicazione non può inviare dati non richiesti, ma deve attendere che uIP chiama l'applicazione. uIP chiama l'applicazione non solo quando arrivano nuovi dati sulla connessione, ma anche come parte dell'elaborazione periodica. Ciò offre all'applicazione la possibilità di inviare dati anche quando non arrivano dati sulla connessione. Le applicazioni che utilizzano UDP possono scegliere di inviare dati in qualsiasi momento e non devono attendere che uIP li chiama.

L'elaborazione dell'output TCP inizia quando l'applicazione è stata chiamata come parte dell'elaborazione dell'input o durante l'elaborazione periodica. Se chiamato come parte dell'elaborazione dell'input, uIP consegna un pacchetto all'applicazione o informa l'applicazione che una connessione è aperta o chiusa. In entrambi i casi, l'applicazione potrebbe voler inviare dati all'host remoto. In tal caso, uIP aggiorna lo stato della connessione per la connessione TCP, aggiunge le intestazioni necessarie ai dati, calcola i checksum necessari e invia il pacchetto. Lo stato della connessione deve essere aggiornato perché i dati dell'applicazione aumenteranno il numero di sequenza TCP per la connessione. L'applicazione potrebbe anche voler chiudere o interrompere la connessione e uIP risponderà di conseguenza.

L'elaborazione dell'output UDP può iniziare quando uIP ha chiamato un'applicazione per i dati in arrivo o perché l'applicazione chiama uIP direttamente. uIP aggiunge l'intestazione UDP ai dati dell'applicazione, con i numeri di porta UDP necessari e calcola il checksum UDP prima di

passare il pacchetto al livello IP per aggiungere l'intestazione IP. Il livello IP aggiunge la sua intestazione e calcola il checksum IP. Il pacchetto viene quindi inviato dal driver del dispositivo di comunicazione.

Periodic Processing

- The purpose of the uIP periodic processing is to update timer-based counters, perform retransmissions, and remove connections that have timed out
- Periodic processing is typically invoked once or twice every second, depending on the configuration of the system in which uIP runs
- Periodic processing starts with updating the status of the IP fragment reassembly buffer
 - If a packet is waiting to be reassembled, the periodic code updates the counter that keeps track of the age of the packet. If the packet is older than 30 seconds, the packet is removed from the reassembly buffer.

Lo scopo dell'elaborazione periodica uIP è aggiornare i contatori basati sul timer, eseguire ritrasmissioni e rimuovere le connessioni scadute.

L'elaborazione periodica viene in genere invocata una o due volte al secondo, a seconda della configurazione del sistema in cui viene eseguito uIP. L'elaborazione periodica inizia con l'aggiornamento dello stato del buffer di riassemblaggio del frammento IP. Se un pacchetto è in attesa di essere riassemblato, il codice periodico aggiorna il contatore che tiene traccia dell'età del pacchetto. Se il pacchetto è più vecchio di 30 secondi, il pacchetto viene rimosso dal buffer di riassemblaggio.

Periodic Processing (cont.)

- Next, the periodic processing code goes through every active TCP connection to check if there are any packets that should be retransmitted
 - If there is a pending retransmission, and if the retransmission includes application data, uIP calls the application to reproduce the data it previously produced for the original transmission of the packet
 - The application, or an application library provided by uIP, may have buffered the packet in an external memory buffer in preparation for a retransmission, in which case the application can copy the buffered packet back to uIP
 - To save memory, however, the application may have chosen not to buffer the packet, but to regenerate it instead. The contents of the packet may, for example, have originated in a ROM buffer from which it can be quickly copied into the retransmission packet, or the contents of the packet are easily regenerated
- In either case, once the application has produced its packet, uIP constructs the necessary headers and sends out the retransmission

Successivamente, il codice di elaborazione periodica passa attraverso ogni connessione TCP attiva per verificare se ci sono pacchetti che devono essere ritrasmessi. Se è presente una ritrasmissione in sospeso e se la ritrasmissione include dati dell'applicazione, uIP chiama l'applicazione per riprodurre i dati precedentemente prodotti per la trasmissione originale del pacchetto. L'applicazione, o una libreria dell'applicazione fornita da uIP, potrebbe aver bufferizzato il pacchetto in un buffer di memoria esterno in preparazione di una ritrasmissione, nel qual caso l'applicazione può copiare il pacchetto bufferizzato su uIP. Per risparmiare memoria, tuttavia, l'applicazione potrebbe aver scelto di non bufferizzare il pacchetto, ma invece di rigenerarlo. Il contenuto del pacchetto può, ad esempio, essere stato originato in un buffer ROM dal quale può essere rapidamente copiato nel pacchetto di ritrasmissione oppure i contenuti del pacchetto possono essere facilmente rigenerati. In entrambi i casi, una volta che l'applicazione ha prodotto il suo pacchetto, uIP costruisce le intestazioni necessarie e invia la ritrasmissione.

Periodic Processing (cont.)

- uIP also updates the retransmission timer by doubling its value, the so-called exponential back-off procedure of TCP
- The periodic TCP code also checks for connections that should be timed out
 - Connections that have retransmitted too many packets without receiving an acknowledgment are examples of this. Such connections are discarded by uIP

uIP aggiorna anche il timer di ritrasmissione raddoppiando il suo valore, la cosiddetta procedura di back-off esponenziale di TCP.

Il codice TCP periodico controlla anche le connessioni che devono essere scadute. Le connessioni che hanno ritrasmesso troppi pacchetti senza ricevere un riconoscimento ne sono un esempio. Tali connessioni vengono scartate da uIP.

Packet Forwarding

- Packet forwarding is done when uIP receives a packet that has a destination IP address that does not match any of the IP addresses of the node
 - A node typically has multiple addresses: one or more unicast addresses and at least one broadcast or multicast address
- Packets that do not match the addresses should be forwarded to a neighboring node, either because the address matches that of the neighbor or because the neighbor has a route to the destination address
- Packet forwarding occurs only when uIP has been configured to be a router. The packet forwarding mechanism is then invoked as part of the output processing
- The packet forwarding mechanism is modular and does not specify any particular routing mechanism to be used.

L'inoltro di pacchetti viene eseguito quando uIP riceve un pacchetto con un indirizzo IP di destinazione che non corrisponde a nessuno degli indirizzi IP del nodo. Un nodo in genere ha più indirizzi: uno o più indirizzi unicast e almeno un indirizzo broadcast o multicast. I pacchetti che non corrispondono agli indirizzi devono essere inoltrati a un nodo adiacente, perché l'indirizzo corrisponde a quello del vicino o perché il vicino ha un percorso verso l'indirizzo di destinazione.

L'inoltro dei pacchetti si verifica solo quando uIP è stato configurato per essere un router. Il meccanismo di inoltro dei pacchetti viene quindi richiamato come parte dell'elaborazione dell'output.

Packet Routing

- A routing mechanism will register itself with the forwarding module upon startup
- For every packet, the forwarding mechanism asks the routing module to look up the destination IP address and return the address to the next-hop neighbor
 - The routing module may implement this any way it wants by using a table of destination addresses, a table of network prefixes, a hash table of addresses, a cache of the recently used routes, or any other way it finds suitable. The routing protocol may perform a route discovery for each address not found in its cache

Il meccanismo di inoltro dei pacchetti è modulare e non specifica alcun meccanismo di routing particolare da utilizzare. Piuttosto, un meccanismo di routing si registrerà automaticamente con il modulo di inoltro all'avvio. Per ogni pacchetto, il meccanismo di inoltro chiede al modulo di routing di cercare l'indirizzo IP di destinazione e restituire l'indirizzo al vicino hop successivo. Il modulo di routing può implementarlo come desidera utilizzando una tabella di indirizzi di destinazione, una tabella di prefissi di rete, una tabella di indirizzi hash, una cache delle route utilizzate di recente o qualsiasi altro modo ritenga opportuno. Il protocollo di routing può eseguire un rilevamento di route per ciascun indirizzo non trovato nella sua cache.

Packet Forwarding and Packet Routing

- By separating packet forwarding and packet routing, uIP can adapt a wide range of requirements such as routing performance and memory requirements, as well as take advantage of future development in routing protocols
- A system with strict memory requirements and low routing performance requirements may use a cache configuration that prompts frequent network route discoveries, whereas a system with strict requirements on routing performance but lax memory requirements may choose a larger cache setting

Separando l'inoltro dei pacchetti e l'instradamento dei pacchetti, uIP può adattare una vasta gamma di requisiti come prestazioni di instradamento e requisiti di memoria, nonché sfruttare i futuri sviluppi dei protocolli di instradamento. Un sistema con requisiti di memoria rigidi e requisiti di prestazioni di routing bassi può utilizzare una configurazione di cache che richiede frequenti rilevamenti di route di rete, mentre un sistema con requisiti rigorosi di prestazioni di routing ma requisiti di memoria bassi possono scegliere un'impostazione di cache più ampia.

uIP Memory Buffer Management

- For smart objects, where data rates typically are much lower than for general purpose computers, **the buffer management strategy does not need to be optimized for high throughput**
- Instead, memory is a scarce resource so the buffer management mechanism must work efficiently with **small amounts of memory**

La gestione del buffer è un'operazione critica in qualsiasi stack. I pacchetti di dati in entrata e in uscita sono bufferizzati in memoria e il sistema di gestione del buffer garantisce che sia disponibile memoria sufficiente per i pacchetti di dati. In un protocollo per scopi generici, una cattiva strategia di gestione del buffer può portare a prestazioni non ottimali. In un oggetto intelligente, dove i requisiti di memoria sono eccezionalmente rigidi, la gestione del buffer ha una funzione fondamentale per garantire che lo stack protocollare sia in grado di funzionare anche quando la memoria è scarsa.

Per fornire un throughput elevato, gli stack IP tradizionali utilizzano strategie di gestione del buffer di varia complessità. I buffer devono essere allocati e deallocati rapidamente per tenere il passo con le grandi quantità di dati provenienti dalla rete e inviati dalle applicazioni. Per gli oggetti intelligenti, in cui le velocità dei dati sono in genere molto più basse rispetto ai computer per scopi generici, non è necessario ottimizzare la strategia di gestione del buffer per un throughput elevato. Al contrario, la memoria è una risorsa scarsa, quindi il meccanismo di gestione del buffer deve funzionare in modo efficiente con piccole quantità di memoria.

uIP Memory Buffer Management (cont.)

- The buffer management strategy of uIP is intentionally simple
 - To keep memory size and code complexity down, all packets in uIP are kept in a **single memory buffer**. Incoming packets are copied to this buffer when the communication device driver has received them. Outgoing packets are created directly into the same buffer
- Using a single memory buffer has several advantages
 - First, there is no need for any complex buffer management mechanisms to be implemented
 - Second, the protocol implementations become simpler when they do not need to deal with multiple buffers
 - Third, because the buffer is at a single place in memory and does not move, the C compiler is often able to make better optimizations at the machine code level, which leads to more efficient use of the scarce code memory

La strategia di gestione del buffer di uIP è intenzionalmente semplice. Per mantenere basse le dimensioni della memoria e la complessità del codice, tutti i pacchetti in uIP sono conservati in un singolo buffer di memoria. I pacchetti in arrivo vengono copiati in questo buffer quando il driver del dispositivo di comunicazione li ha ricevuti. I pacchetti in uscita vengono creati direttamente nello stesso buffer.

L'uso di un singolo buffer di memoria presenta numerosi vantaggi. Innanzitutto, non è necessario implementare meccanismi complessi di gestione dei buffer. Tali meccanismi richiedono spazio di codice e memoria buffer, entrambi vantaggiosi in un sistema con memoria limitata. In secondo luogo, le implementazioni del protocollo diventano più semplici quando non devono gestire più buffer. Terzo, poiché il buffer si trova in un unico punto della memoria e non si sposta, il compilatore C è spesso in grado di effettuare migliori ottimizzazioni a livello di codice macchina, il che porta a un uso più efficiente della memoria.

uIP Memory Buffer Management (cont.)

- The packet buffer is large enough to contain one packet of maximum size
- When a packet arrives from the network, the device driver places it in the global buffer and calls the uIP input processing code
 - If the packet contains application data, uIP calls the corresponding application with the application data in the packet buffer. Because the data in the buffer will be overwritten by the next incoming packet, the application will either have to act immediately on the data or copy the data into a secondary buffer for later processing
 - The packet buffer will not be overwritten by new packets before the application has finished processing the data.
- To ensure that the packet buffer is not overwritten while uIP is processing a packet, uIP does not allow the communication device driver to write directly into the buffer, except when uIP explicitly asks it to do so
 - If a packet arrives when uIP is processing data in the buffer, the packet is queued either in the hardware of the communication device or by the communication device driver. Most communication devices have a hardware-implemented buffer memory in which they store incoming packets as they arrive

Il buffer dei pacchetti è abbastanza grande da contenere un pacchetto di dimensioni massime. Quando un pacchetto arriva dalla rete, il driver del dispositivo lo inserisce nel buffer globale e chiama il codice di elaborazione dell'input uIP. Se il pacchetto contiene dati dell'applicazione, uIP chiama l'applicazione corrispondente con i dati dell'applicazione nel buffer dei pacchetti. Poiché i dati nel buffer verranno sovrascritti dal pacchetto in entrata successivo, l'applicazione dovrà agire immediatamente sui dati o copiarli in un buffer secondario per l'elaborazione successiva. Il buffer dei pacchetti non verrà sovrascritto da nuovi pacchetti prima che l'applicazione abbia terminato l'elaborazione dei dati.

Per garantire che il buffer dei pacchetti non venga sovrascritto mentre uIP elabora un pacchetto, uIP non consente al driver del dispositivo di comunicazione di scrivere direttamente nel buffer, tranne quando uIP gli chiede esplicitamente di farlo. Se arriva un pacchetto quando uIP elabora i dati nel buffer, il pacchetto viene messo in coda nell'hardware del dispositivo di comunicazione o dal driver del dispositivo di comunicazione. La maggior parte dei dispositivi di comunicazione dispone di una memoria buffer implementata dall'hardware in cui memorizzano i pacchetti al loro arrivo.

La dimensione del buffer dei pacchetti è configurabile al momento della compilazione. La quantità totale di utilizzo della memoria per uIP dipende dalle applicazioni del particolare sistema in cui devono essere eseguite le implementazioni. La configurazione della memoria determina sia la quantità di traffico che il sistema dovrebbe essere in grado di gestire sia la quantità massima di connessioni

simultanee. Un sistema che invia e riceve grandi quantità di dati a più client simultanei è in genere configurato per utilizzare più memoria di un sistema che occasionalmente invia pochi byte. È possibile eseguire UIP con un minimo di 100 byte di RAM, ma tale configurazione fornisce un throughput estremamente basso e consente solo un numero limitato di connessioni simultanee.

uIP Protocol Implementations

Per ottimizzare la memoria e lo spazio del codice, le implementazioni del protocollo uIP implementano solo i meccanismi necessari per la conformità agli standard e l'interoperabilità. Molti dei meccanismi nella suite di protocolli TCP / IP sono stati progettati per migliorare le prestazioni del protocollo. In generale, uIP non implementa tali meccanismi che forniscono prestazioni migliori a scapito di requisiti di memoria più elevati.

IP Fragmente Reassembly

- IP fragment reassembly is implemented using a separate buffer that holds the packet to be reassembled
- An incoming fragment is copied into the right place in the buffer and a bit map is used to keep track of which fragments have been received
 - Because the first byte of an IP fragment is aligned on an 8-byte boundary, the bit map requires a small amount of memory
- When all fragments have been reassembled, the resulting IP packet is passed to the transport layer
- If all fragments have not been received within a specified time frame, the packet is dropped
- uIP maintains a single buffer for holding packets to be reassembled
 - Because uIP only maintains a single buffer for reassembling fragments, uIP does not support simultaneous reassembly of more than one packet. The reason for this design decision is that IP fragments are relatively uncommon in today's networks

Il riassemblaggio del frammento IP viene implementato utilizzando un buffer separato che contiene il pacchetto da riassemblare. Un frammento in arrivo viene copiato nel posto giusto nel buffer e una bit map viene utilizzata per tenere traccia di quali frammenti sono stati ricevuti. Poiché il primo byte di un frammento IP è allineato su un limite di 8 byte, la bit map richiede una piccola quantità di memoria. Quando tutti i frammenti sono stati riassemblati, il pacchetto IP risultante viene passato al livello di trasporto. Se tutti i frammenti non sono stati ricevuti entro un intervallo di tempo specificato, il pacchetto viene eliminato.

uIP mantiene un unico buffer per contenere i pacchetti da riassemblare. Il buffer è separato dal buffer dei pacchetti utilizzato dal test di uIP per evitare di essere sovrascritto da altri pacchetti. Poiché uIP mantiene solo un singolo buffer per riassemblare i frammenti, uIP non supporta il riassemblaggio simultaneo di più di un pacchetto. Il motivo di questa decisione di progettazione è che i frammenti IP sono relativamente rari nelle reti odierne.

TCP

There are several mechanisms in TCP intended to provide a high throughput. Many of these mechanisms are not needed in a system that has only small amounts of data to be sent

uIP makes the trade-off that **memory efficiency is more important than high data throughput**

L'implementazione TCP in uIP è progettata per essere il più semplice possibile senza rimuovere nessuno dei meccanismi TCP richiesti. Esistono diversi meccanismi in TCP destinati a fornire un throughput elevato. Molti di questi meccanismi non sono necessari in un sistema che ha solo piccole quantità di dati da inviare. uIP quindi sceglie che come compromesso l'efficienza della memoria è più importante dell'elevata velocità di trasmissione dei dati. Se è richiesto un throughput di dati elevato, uIP non è una scelta adatta.

L'implementazione TCP in uIP è guidata dai pacchetti in entrata e dall'elaborazione periodica. I pacchetti in arrivo vengono analizzati da TCP e se il pacchetto contiene dati da consegnare all'applicazione, l'applicazione viene invocata dalla chiamata della funzione dell'applicazione. Se il pacchetto in entrata riconosce i dati precedentemente inviati, lo stato della connessione viene aggiornato e l'applicazione viene informata, consentendogli di inviare nuovi dati.

TCP consente a una connessione di ascoltare le richieste di connessione in arrivo. In uIP, una connessione in ascolto è identificata dal numero di porta a 16 bit e le richieste di connessione in entrata vengono verificate rispetto all'elenco delle connessioni in ascolto. Questo elenco di connessioni di ascolto è dinamico e può essere modificato dalle applicazioni nel sistema.

Sliding Window

- Most TCP implementations use a sliding window mechanism when sending data. This is intended to provide a high throughput
- An implementation of the sliding window mechanism uses a significant amount of 32-bit additions and subtractions because of the 32-bit sequence numbers used by TCP
 - 32-bit arithmetic is expensive regarding code size on many 8- and 16-bit microcontrollers
- uIP does not implement the sliding window mechanism
- Also, uIP does not buffer sent packets
 - A sliding window implementation that does not buffer sent packets will have to be supported by a complex application layer
- Instead, uIP allows only a single TCP segment per connection to be unacknowledged at any given time
- Removing the sliding window mechanism does not affect interoperability or standards compliance in any way

La maggior parte delle implementazioni TCP utilizza un meccanismo a finestra scorrevole durante l'invio di dati. Più segmenti di dati vengono inviati in successione senza attendere un riconoscimento per ogni segmento. Questo ha lo scopo di fornire un throughput elevato perché l'intera banda di rete tra il mittente e il destinatario può essere riempita di pacchetti senza attendere che il destinatario riconosca la ricezione dei pacchetti.

Un'implementazione del meccanismo a finestra scorrevole utilizza una quantità significativa di aggiunte e sottrazioni a 32 bit a causa dei numeri di sequenza a 32 bit utilizzati da TCP. Poiché l'aritmetica a 32 bit è costosa per quanto riguarda la dimensione del codice su molti microcontrollori a 8 e 16 bit, uIP non implementa il meccanismo a finestra scorrevole. Inoltre, uIP non esegue il buffer dei pacchetti inviati e un'implementazione a finestra scorrevole che non esegue il buffer dei pacchetti inviati dovrà essere supportata da un livello applicativo complesso. Al contrario, uIP consente di riconoscere un solo segmento TCP per connessione in qualsiasi momento.

È importante notare che anche se la maggior parte delle implementazioni TCP utilizza l'algoritmo a finestra scorrevole, non è richiesto dalle specifiche TCP. La rimozione del meccanismo a finestra scorrevole non influisce in alcun modo sull'interoperabilità o sulla conformità agli standard.

Retransmission and RTT Estimation

- Retransmissions are driven by the periodic TCP timer
 - Every time the periodic timer is invoked, the retransmission timer for each connection is decremented
 - If the timer reaches zero, a retransmission should be made
- To find a suitable value for the retransmission timeout, TCP continuously estimates the current RTT of every active connection. The RTT estimation in uIP is implemented using TCP's periodic timer
- Each time the periodic timer fires, it increments a counter for each connection that has unacknowledged data in the network. When an acknowledgment is received, the current value of the counter is used as a sample of the RTT

Le ritrasmissioni sono guidate dal timer TCP periodico. Ogni volta che viene richiamato il timer periodico, il timer di ritrasmissione per ogni connessione viene ridotto. Se il timer raggiunge lo zero, è necessario effettuare una ritrasmissione.

Per trovare un valore adatto per il timeout di ritrasmissione, TCP stima continuamente l'attuale RTT di ogni connessione attiva. La stima RTT in uIP viene implementata utilizzando il timer periodico TCP. Ogni volta che il timer periodico si attiva, aumenta un contatore per ogni connessione che ha dati non riconosciuti nella rete. Quando viene ricevuto un ACK, il valore corrente del contatore viene utilizzato come campione di RTT. Questo esempio viene utilizzato insieme alla funzione di stima RTT TCP standard di Van Jacobson per calcolare una stima dell'RTT. L'algoritmo Karn e Partridge viene utilizzato per garantire che le ritrasmissioni non distorcano le stime.

Flow Control

- The purpose of the TCP flow control mechanisms is to allow communication between hosts with wildly varying memory dimensions
- In each TCP segment, the sender of the segment indicates its available buffer space
- A TCP sender must not send more data than the buffer space indicated by the receiver.
- In uIP, the application cannot send more data than the receiving host can buffer, and an application cannot send more data than the amount of bytes it is allowed to send by the receiving host. If the remote host cannot accept any data at all, the stack initiates the zero-window probing mechanism

Lo scopo dei meccanismi di controllo del flusso TCP è quello di consentire la comunicazione tra host con dimensioni della memoria estremamente variabili. In ogni segmento TCP, il mittente del segmento indica il suo spazio buffer disponibile. Un mittente TCP non deve inviare più dati dello spazio buffer indicato dal destinatario.

In uIP, l'applicazione non può inviare più dati di quanti ne possa bufferizzare l'host ricevente e un'applicazione non può inviare più dati della quantità di byte che può inviare l'host ricevente. Se l'host remoto non può accettare alcun dato, lo stack avvia il meccanismo di rilevamento a finestra zero.

Congestion Control

- The congestion control mechanisms limit the number of simultaneous TCP segments in the network
- Since uIP only handles one in-flight TCP segment per connection, the amount of simultaneous segments cannot be further limited, thus **the TCP congestion control mechanisms are not needed**

I meccanismi di controllo della congestione limitano il numero di segmenti TCP simultanei nella rete. Fin dall'inizio, gli algoritmi utilizzati per il controllo della congestione sono progettati per essere semplici da implementare e la loro implementazione richiede solo poche righe di codice.

Poiché uIP gestisce solo un segmento TCP per connessione, la quantità di segmenti simultanei non può essere ulteriormente limitata, quindi i meccanismi di controllo della congestione TCP non sono necessari.

Urgent Data

- TCP's urgent data mechanism provides an application-to-application notification mechanism, which can be used by an application to mark parts of the data stream as more urgent than the normal stream. It is up to the receiving application to interpret the meaning of the urgent data.
- In many TCP implementations, including the BSD implementation, the urgent data feature increases the complexity of the implementation because it requires an asynchronous notification mechanism in an otherwise synchronous API
- As uIP already uses an asynchronous event-based API, the implementation of the urgent data feature does not lead to increased complexity

Il meccanismo di dati urgenti di TCP fornisce un meccanismo di notifica da applicazione a applicazione, che può essere utilizzato da un'applicazione per contrassegnare parti del flusso di dati come più urgenti del flusso normale. Spetta all'applicazione ricevente interpretare il significato dei dati urgenti.

In molte implementazioni TCP, inclusa l'implementazione di BSD, la funzione di dati urgenti aumenta la complessità dell'implementazione perché richiede un meccanismo di notifica asincrono in un'API altrimenti sincrona. Poiché uIP utilizza già un'API asincrona basata su eventi, l'implementazione della funzione di dati urgenti non comporta una maggiore complessità.

Checksum Calculation

- The TCP and IP protocols implement a checksum that covers the data and header portions of the TCP and IP packets. Since the calculation of this checksum is made over all bytes in every packet sent and received, it is important that the function that calculates the checksum is efficient
- This means that the checksum calculation must be fine-tuned for the particular architecture on which the uIP stack runs

I protocolli TCP e IP implementano un checksum che copre i dati e le porzioni di intestazione dei pacchetti TCP e IP. Poiché il calcolo di questo checksum viene eseguito su tutti i byte in ogni pacchetto inviato e ricevuto, è importante che la funzione che calcola il checksum sia efficace. Molto spesso, ciò significa che il calcolo del checksum deve essere perfezionato per la particolare architettura su cui viene eseguito lo stack uIP. Mentre uIP include una funzione di checksum generica, è anche aperta a un'implementazione specifica dell'architettura delle due funzioni `uip_ipchksum ()` e `uip_tcpchksum ()`. I calcoli di checksum in tali funzioni possono essere scritti in assemblatore altamente ottimizzato anziché in codice C generico.

Memory Footprint

Function	ROM	RAM
IPv6 ND input/output	4800	20
IPv6 ND structures	2128	238
IPv6 network interface management	1348	118
IPv6 address autoconfiguration	372	16
IPv6 input processing	1434	44
Packet buffer	0	1296
UDP	1345	0
TCP	4192	240
Total	15,619	1972

Note: The data are given in bytes. As a comparison, the memory footprint IP stack in a modern general purpose operating system is several hundreds of thousands of bytes.

Memory Footprint of the Individual Functions in the uIPv6 Stack

L'impronta di memoria di uIP è molto ridotta rispetto agli stack IP esistenti per computer di uso generale. Lo stack IP in Linux, ad esempio, richiede diverse centinaia di migliaia di byte di memoria. Per i microcontrollori con memoria limitata utilizzati negli oggetti intelligenti, tale stack IP non si adatta.

L'impronta del codice di uIP è di pochi kilobyte e l'impronta della memoria è inferiore a 2 kB. L'impronta del codice è leggermente superiore per IPv6 rispetto a IPv4. La tabella mostra la suddivisione del codice e del footprint di memoria per le diverse funzioni in uIPv6. L'impronta viene misurata per il processore Atmel128 Atmel e il codice viene compilato con il compilatore C gcc.

Come mostrato nella tabella, circa metà dell'impronta del codice viene utilizzata da IPv6 e il resto da TCP e UDP. Per IPv6, l'ND è la parte più grande, mentre l'elaborazione dell'input IPv6 è piccola.

Conclusions

- A common belief has been that the IP stack is too complex to efficiently implement for the microcontrollers used in smart objects. This belief has been shown to be false by several memory-efficient implementations of the IP stack
- In 2008, uIP was the first IP stack for smart objects to be certified under the IPv6 Ready program
- uIP implements the most important protocols in the IP stack: IP, ICMP, UDP, and TCP. It contains both an IPv4 and an IPv6 implementation
- The code and memory footprint of uIP is very small compared to that of IP stacks for general purpose operating systems. uIP requires only a few kilobytes of code to implement a full IPv6 stack. To achieve such a small footprint, the design of uIP is intentionally simple
 - Packets are processed in sequence
 - The buffer management mechanism is simple
 - The application API is event-driven
- The uIP stack is not unique in its efficient memory. The mechanisms from uIP have been adopted by several other IP stacks for smart objects

Una convinzione comune è che lo stack IP sia troppo complesso per essere implementato in modo efficace per i microcontrollori utilizzati negli oggetti intelligenti. Questa convinzione ha dimostrato di essere falsa da diverse implementazioni efficienti della memoria dello stack IP. Abbiamo studiato lo stack uIP open source ampiamente utilizzato, il primo stack IP per oggetti intelligenti, per vedere come raggiunge il suo codice basso e il suo ingombro di memoria. uIP è stato rilasciato per la prima volta nel 2001. Nel 2008, uIP è stato il primo stack IP per oggetti intelligenti ad essere certificato nell'ambito del programma IPv6 Ready. uIP implementa i protocolli più importanti nello stack IP: IP, ICMP, UDP e TCP. Contiene sia un'implementazione IPv4 che un'implementazione IPv6. Protocolli a livello di applicazione come HTTP e SNMP sono implementati su uIP. Il footprint di codice e memoria di uIP è molto ridotto rispetto a quello degli stack IP per sistemi operativi generici. uIP richiede solo pochi kilobyte di codice per implementare uno stack IPv6 completo. Per ottenere un ingombro così ridotto, il design di uIP è intenzionalmente semplice. I pacchetti vengono elaborati in sequenza, il meccanismo di gestione del buffer è semplice e l'API dell'applicazione è guidata dagli eventi. Lo stack uIP non è unico nella sua memoria efficiente. I meccanismi di uIP sono stati adottati da molti altri stack IP per oggetti intelligenti.